An Anti-Malware Product Test Orchestration Solution for Multiple Pluggable Environments

UNIVERSITY OF TURKU
Department of Future Technologies
Master of Science in Technology Thesis
Networked Systems Security
November 2018
Habibul Islam

Supervisor:
Bruno Amaro Almeida (F-Secure)

Examiners:
Seppo Virtanen
Antti Hakkala

UNIVERSITY OF TURKU
Department of Future Technologies

HABIBUL ISLAM: An Anti-Malware Product Test Orchestration Solution for Multiple Pluggable Environments

Master of Science in Technology Thesis, 62 pages.
Networked Systems Security
November 2018

The term automation gets thrown around a lot these days in the software industry. However, the recent change in test automation in the software engineering process is driven by multiple factors such as environmental factors, both external and internal as well as industry-driven factors. Simply, what we all understand about automation is - the use of some technologies to operate a task. The choice of the right tools, be it in-house or any third-party software, can increase effectiveness, efficiency and coverage of the security product testing.

Often, test environments are maintained at various stages in the testing process. Developer's test, dedicated test, integration test and pre-production or business readiness test are some common phrases in software testing. On the other hand, abstraction is often included between different architectural layers, ever-changing providers of virtualization platforms such as VMWare, OpenStack, AWS as test execution environments and many others with a different state of maintainability. As there is an obvious mismatch in configuration between development, testing and production environment; software testing process is often slow and tedious for many organizations due to the lack of collaboration between IT Operations and Software Development teams. Because of this, identifying and addressing test environment-related compatibility becomes a major concern for QA teams.

In this context, this thesis presents a DevOps approach and implementation method of an automated test execution solution named OneTA that can interact with multiple test environments including isolated malware test environments. The study was performed to identify a common way of preparing test environments in in-house and publicly available virtualization platforms where distributed tests can run on a regular basis. The current solution allows security product testing in multiple pluggable environments in a single setup utilizing the modern DevOps practice to result minimum efforts.

This thesis project was carried out in collaboration with F-Secure, a leading cyber security company in Finland. The project deals with the company's internal environments for test execution. It explores the available infrastructures so that software development team can use this solution as a test execution tool.

Keywords: Test Automation, Continuous Integration, DevOps, Automated Malware Testing, Python

Table of Contents

1 Introduction.. 1

1.1 Background 3

1.2 Aims and objectives 4

1.3 Research outline 6

1.4 Thesis structure 7

2 Test Automation and DevOps Practice 9

2.1 Defination of DevOps 9

2.2 The rise of Agile and DevOps 10

2.3 The key areas of DevOps 12

2.4 Test automation in DevOps ecosystem 13

2.5 Automation in testing 14

2.6 DevOps tools and technological solutions 15

2.7 Automation framework Ansible 17

2.8 Virtualization in Test Automation 18

3 Overview of the Target Environments 19

3.1 Pluggable test environments 19

3.1.1 Bare-metal environment 20

3.1.2 Dynamic Virtual Machine Provisioning Systems (DVMPS) 21

3.1.3 AWS EC-2 23

3.1.4 Sandbox solution 25

3.1.5 Isolated malware test environment: Red Cloud 26

4 Defining the Test Related Requirements and Problems............... 30

4.1 Specifying the functionality 30

4.2 Infra instability and inconsistency 31

4.3 Future consideration for a widespread range of cloud platforms 31

4.4 Security considerations related to handling real malware 32

4.5 Main requirements for OneTA solution 32

5 Design and Implementation of the Orchestration Solution.......... 34

5.1 Logical architecture and main workflow of OneTA solution 34

5.2 Fragmentation of tools and required resources 35

5.3 Backend and client-side infrastructures 38

5.4 Test machine creation in DVMPS environment 39

5.5 Inclusion of in-house remote execution tool “FSExec” 40
5.6 Python Paramiko for SSH connectivity 41
5.7 REST API for the Sandbox solution 42
5.8 Ansible for AWS configuration management 43
5.9 JSON to YAML conversion 44
5.10 JSON to manage test configuration in Python 45
5.11 Automating workflow with Python 47
5.12 Jenkins for continuous integration 49
5.13 Running the Tests 49
5.14 Packer for machine image creation 53

6 Results: Justification of the Proof of Concept 54

6.1 Justification of requirements 54
6.2 Validity, Reliability and Stability of the concept 56
6.3 Test cases 56
6.4 Test coverage and core features 57

7 Conclusion 59

7.1 Limitations and suggestions for future work 61

References 63

List of Figures

Figure 1: The principal of OneTA test orchestration solution 5
Figure 2: Research workflow of the OneTA solution 6
Figure 3: General overview of DevOps in the software development lifecycle 11
Figure 4: Manual steps for creating VM in DVMPS environment 23
Figure 5: Overview of the Red cloud architecture and purposes of labeling 28
Figure 6: Actual system design and principal of workflow of the OneTA solution 34
Figure 7: Life cycle of OneTA in anti-malware product test use cases 52

List of Tables

Table 1: List of commonly used DevOps tools 15
Table 2: List of target environments for conducting the tests 20
Table 3: Purposes and labeling of different environments inside the organization 27
Table 4: List of OneTA components and roles in the test process 38
Table 5: Sandbox REST API use cases 42
Table 6: List of parameters required for running the test 51
Table 7: Currently available supports for test orchestration through OneTA 55

Code Snippets

Code Snippet 1: Code snippet for creating Ansible playbook using JSON values 45
Code Snippet 2: Example of JSON for AWS use case 46
Code Snippet 3: Implementation of command-line interface using docopt 48

Abbreviations and Acronyms

AV	Anti-Virus
AWS	Amazon Web Services
AMI	Amazon Machine Image
API	Application Programming Interface
CI	Continuous Integration
CD	Continuous Delivery
CLI	Command-line interface
CPU	Central Processing Unit
EC2	Elastic Compute Cloud
EICAR	European Institute for Computer Antivirus Research
GUI	Graphical User Interface
GNU	General Public License
HW	Hardware
IT	Information Technology
IaaS	Infrastructure-as-a-Service
JSON	JavaScript Object Notation
KVM	Kernel-based Virtual Machine
MIT	Massachusetts Institute of Technology
PyPI	Python Package Index
PoC	Proof of Concept
QA	Quality Assurance
SSH	Secure Shell
SSL	Secure Sockets Layer
SHA	Secure Hash Algorithm
SHA1	Secure Hash Algorithm 1
SCM	Source Code Management
TA	Test Automation
TCP	Transmission Control Protocol
TEM	Test Environment Management
UDP	User Datagram Protocol
URL	Uniform Resource Locator
VPN	Virtual Private Network
VM	Virtual Machine
VCS	Version control systems
WWW	World Wide Web
YAML	YAML Ain't Markup Language

1 Introduction

We all know that computers are wonderful machines. They give us the power to accomplish anything that we want these days. They can be taught to perform many tasks in a time effective way. Over the last several years, there have been significant advances in the adoption of new automation technologies in IT industries. The main reason for this is to accelerate the ongoing digital transformation.

In any professional endeavor, people usually deal with different kinds of systems. Being able to deal with different computer systems not only entails knowing what specific requirements need to be fulfilled but it also entails having the ability to think like a computer. Significantly, most of the software development teams demand an effective and secure test execution process where reliable tests can run on a daily basis. However, the execution process often relies on multiple environments and they are often distinctive. Real-time remote management and simplified edge infrastructure are pivotal where more data-intensive computing workload is involved. In this context, the power to make this bidding for us is the appropriate implementation of "Programming Paradigm" which can make the compatibility to accomplish our needs programmatically.

Nowadays test automation (also called as TA) is fundamental in the agile development context. By adopting the automated testing approach, we can speed up the process of software validation and increase test coverage. It has become commonplace in the field of malware analysis and development of anti-malware software to perform the software testing programmatically. However, there are many challenges in applying test automation for applications under validation [2]. In any security software development process, the malware handling needs to be automated but secure. Moreover, the test environment needs to be able to execute malware without allowing it to escape to other computers and networks. An effective and efficient management of anti-malware product test environments with structured execution process can deliver significant benefits and bring down the walls between the teams and align incentives through automation, lean principles and measurement practices. Many test automation frameworks are available, and they supply different purposes in the software testing process. A framework can be defined in many ways and there are various definitions available for it. However, in test automation domain it can be defined as such:

"A test automation framework is a collection of interacting components facilitating the creation and execution of automated tests and the reporting of the results thereof." [8]

The term automation comes into force when we need to deal with repetitive tasks. Test automation can automate some repetitive tasks and it is critical for continuous delivery (CD) and continuous testing. In this context, testing of anti-malware software in an automated form requires an end-to-end secure connection, where simulated malware samples and infections are heavily involved. Additionally, testing real malware adds many requirements for the test environment and infrastructure. To ensure the effectiveness of any anti-malware software, automated functional testing in different systems is necessary. Unfortunately, there is no comprehensive generic solution available for it [30]. Many automation tools and frameworks are available in the market, but it is hard to find an absolute support that we oftentimes require in our systems. Anti-malware vendors and security research teams often need to implement their own testing solution to support multiple environments they use. To mitigate the potential drawbacks and obtain ultimate advantages, it is reasonable to take a hybrid approach for testing the software on various levels of test target abstraction.

As there was a need to build a new integrated tool or extend existing software engineering tools and design a clear DevOps pipeline, this thesis work seeks to solve an unsolved on-premise IT orchestration challenge by developing a clear automation process of complex multi-tier workflows under a single banner. This thesis presents an approach and implementation method of an automated test execution solution named OneTA, which is mainly a collection of Python scripts for distributed execution of automated anti-malware product tests in multiple environments. However, the scripts enable the reuse of functions, test scenarios, and the collection of user actions, which results in less effort. The proposed approach, consists of two main elements; a controller machine, and test infrastructures. The project aimed to design and implement a secure solution that satisfies the main requirements for automated network connectivity for different test environments used by this thesis commissioning software vendor. The project also deals with totally isolated cloud-based malware test environment and provide a significant solution to execute the test as a pluggable test environment. Furthermore, it enables various stakeholders of the test execution domain to perform the test in a Continuous Integration (CI) method and deal with time-consuming and repetitive tasks.

However, the underlying meaning behind OneTA is "One Test Automation" and this naming was inspired from another solution of this thesis commissioning organization. OneTA combines a wealth of different tools and technologies, all preconfigured into a single framework for vendor's internal use. Therefore, it allows multiple automation components to provide end-to-end test automation for many test cases. In a nutshell, OneTA can be summarized in the following way:

OneTA is a collection of Python scripts and libraries unified by one namespace that provides a standard set of instructions to access multiple systems simultaneously and interact with them by covering all their dependencies.

1.1 Background

In the test consulting domain, the testers and test managers change domains frequently due to a large set of test cases involved. Virtualization platforms have grown to play an essential role in this change. In this research conducting organization, several test execution environments are in use and they are in different states of maintainability. Similarly, different methods for test execution are available and they are preferred to be done programmatically. On the other hand, a secure way of conducting the tests is always a big challenge, because testing anti-malware products usually performed against real malware samples as well as crafted malware samples. Moreover, some tests in the in-house environments often involve manual tasks. Manual testing is laborious, and it is also a time-consuming process. In this scenario, some common problems faced in existing implementations were identified by the test managers in this test consulting domain prior to their analysis and those problems are outlined below:

- Widespread range of in-house test environments and tools
- Ever changing providers of virtualization platforms such as VMWare, OpenStack, AWS, etc.
- No common provisioning support for multiple test environments
- No clear abstraction between different architectural layers
- No infra stability and consistency
- Many tools and framework in a different state of maintainability
- Manual and poorly maintained crafted template images

1.2 Aims and objectives

The prime goal of this thesis project was to allow development and QA teams to perform product testing against multiple systems using a simple test definition. It also focuses on the broad analysis of techniques, tools and knowledge needed to manage test environments in the software engineering processes and infrastructure automation to solve those problems.

In summary, the objectives which were formed before the study are listed below:

i. A common test automation model needs to be implemented for available services

ii. The solution/library can be used via command-line (standalone) or integrated with other in-house systems

iii. Must support in-house sandboxing solution as a pluggable environment to allow testing against unknown files and URLs

iv. Must support Amazon Web Services (AWS) infrastructure as the pluggable environment

v. Empirical results of this solution should be effective against vendor's Red Test Automation use cases

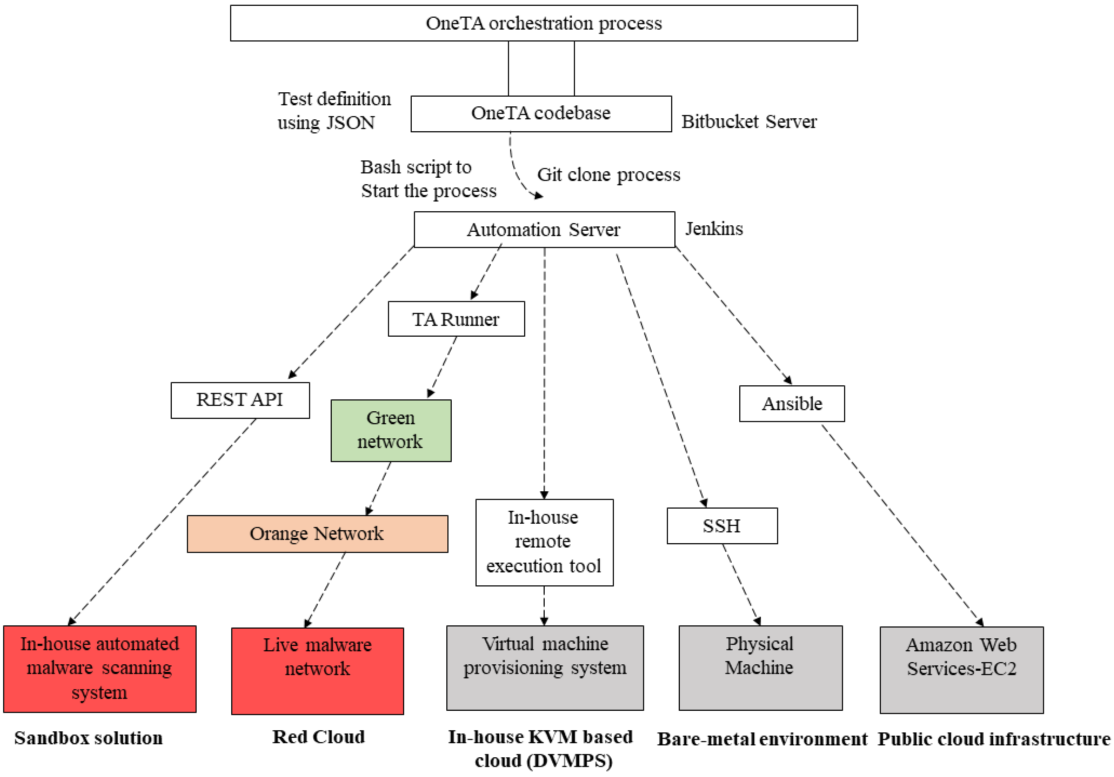
vi. Should have the ability to rerun existing tests on new infrastructure

vii. Must allow measurements such as performance, detection capabilities and other factors in a safe/isolated environment

viii. Able to eliminate errors due to manual interventions and delays due to dependencies

ix. Provide an efficient solution for a security research team in the test execution process using supporting technologies

To give an illustration of the overall concept and requirements, Figure 1 presents the high-level architectural overview of the OneTA solution. For the sake of clarity, those components which are not closely related to OneTA workflow are excluded from figure 1. The actual implementation process will be explained in chapter 5.



OneTA orchestration process
Test definition using JSON
OneTA codebase
Bitbucket Server
Bash script to Start the process
Git clone process
Automation Server
Jenkins
TA Runner
REST API
Green network
Ansible
Orange Network
In-house remote execution tool
SSH
In-house automated malware scanning system
Live malware network
Virtual machine provisioning system
Physical Machine
Amazon Web Services-EC2
Sandbox solution
Red Cloud
In-house KVM based cloud (DVMPS)
Bare-metal environment
Public cloud infrastructure

Figure 1: The principal of OneTA test orchestration solution

The research project, as well as this thesis, is about developing a Python-based library that helps in saving time and reduction of manual intervention. The solution should allow to write, run, and analyze automated tests, except for the tests themselves. Also, it involves a broad analysis of knowing how existing tools can be integrated into this solution and implement a fully functional test automation process to support multiple environments. In this thesis, “development and test” mainly refers to the various tools and industry-driven practices applied when unifying test orchestration process.

1.3 Research outline

This thesis presents a DevOps methodology, which explores the challenge of unification of different tools and execution methods. Additionally, it tries to address the dependency related issues, which were identified by the test managers. The project was very technical in nature, and Python 3 was used as the main scripting language to make the systems functional and operational. Therefore, the system was verified against a few test cases for the functionality. Figure 2 shows the overall research workflows that were followed during the development of the OneTA solution.

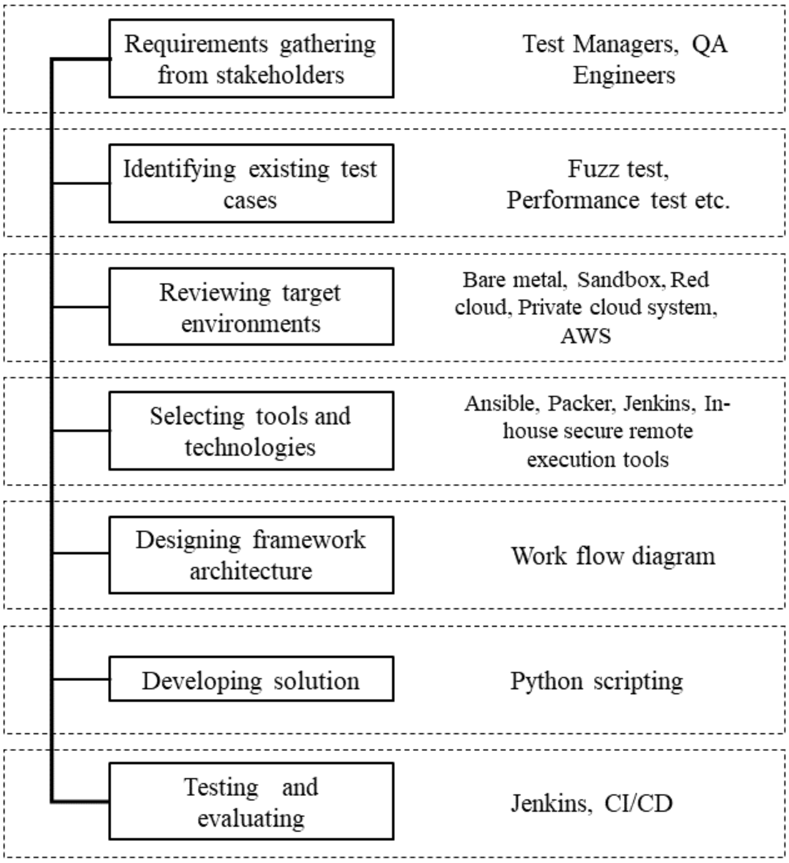

Requirements gathering from stakeholders
Test Managers, QA Engineers
Identifying existing test cases
Fuzz test, Performance test etc.
Reviewing target environments
Bare metal, Sandbox, Red cloud, Private cloud system, AWS
Selecting tools and technologies
Ansible, Packer, Jenkins, In-house secure remote execution tools
Designing framework architecture
Work flow diagram
Developing solution
Python scripting
Testing and evaluating
Jenkins, CI/CD

Figure 2: Research workflow of the OneTA solution

The overall research was conducted concentrating on these areas:

- DevOps methodology
- Automated provisioning
- Configuration automation
- Virtualization
- Anti-malware product testing
- Malware analysis
- Continuous integration

1.4 Thesis structure

In this thesis, the discussion centred on a security product test automation solution that is suited for different test environments with different business requirements along with multiple virtualization platforms. The project thoroughly followed the test automation strategies. This thesis is divided into seven chapters. The primary intent of this chapter was to provide an overall idea of the work and some problem definitions. The rest of the chapters present common terminologies, related technologies, theoretical background, system specification, architectural overview and justification of the proof of concept. This thesis is structured as follows:

Chapter 2 describes the theoretical knowledge of DevOps and the importance of DevOps as a practice in modern software development process. The description of this chapter is based on the knowledge acquired during the thesis study. Additionally, it also outlines some of the commonly used DevOps tools, which are predominantly used in the modern software industries.

Chapter 3 will introduce the target environments for which this solution was developed. It tries to give an overall architecture of the systems that OneTA interact throughout the test orchestration process. It will briefly explain how the internal systems work and their main purpose of use.

Chapter 4 presents the common challenges that are usually faced in the test automation context. It discusses some of the issues which were particularly identified during this study.

Chapter 5 provides a broad description of the different components which were used to develop the solution. It describes the higher-level architecture, supporting technologies and OneTA specific approach. It provides a broad view of the actual solution and required methods needed to fulfil the thesis objectives.

Chapter 6 highlights the actual outcomes of this project and justification of the proof of concepts. It evaluates the objectives that were mentioned in chapter 1. It discusses the overall performance and acceptance of the solution.

Chapter 7 concludes with the solution that has been presented with the corresponding outcomes. It discusses the future possibility of expansion of this solution. It also outlines how we can support more pluggable environments with a similar approach.

2 Test Automation and DevOps Practice

To enable teams to unlock the potential of the modern technologies, especially in the infrastructure and operations realm, many factors are fueling the IT automation in the modern software industry. According to “State of DevOps Report” by Puppet from 2016, uncovered that high-performing IT teams spend 50 percent less time solving security issues [33]. Utilizing DevOps in practice, then developing automation solutions and processes in the security software testing was the main motive of this thesis project. Indeed, it is important to understand why DevOps exists in modern software development lifecycle and what crucial role test automation plays in DevOps ecosystem. Most importantly, why does it seem like most of the companies are moving in the direction of DevOps and how we can get the benefit from it.

The present chapter of this thesis tries to define what DevOps brings to an organization regarding the automation process. The primary objective of this chapter is to give a brief introduction about DevOps and technical perspective of this term in the software development process. It mainly focuses on the importance of DevOps practice and how test automation fits into it. Also, it introduces some DevOps tools and technologies that have greater impact on ongoing IT automation and DevOps practices.

2.1 Defination of DevOps

There has been a significant improvement in test automation in the last few years. As it happens in any growing industry, many trends were set. The trend nowadays in many IT organizations is having a culture shift towards DevOps in their everyday work practices [36]. In the first place, what is known as DevOps? DevOps, the combination of Development and Operation, is a practice that is followed by the software companies for better collaboration, for better results and for building trust among the teams [20] [44]. More specifically, it is a mindset and culture. DevOps is not confined to one tool, or it is not a role, it reduces the unnecessary back and forth issues between teams [37].

Not so long ago, the typical IT story involved highly skilled sysadmins who used to create and maintain the systems manually they were responsible for. These systems were often totally managed by hand and trusted to live a long and productive life. Moreover, there was a heavy division between application developers and the system administrators running the systems that an application would run on. To alleviate these issues, DevOps facilitates application developers and sysadmins to work intently to automate the delivery process [42]. Thus, it minimizes the abstraction between software developers and system administrators who are involved in building applications and keep the infrastructure running respectively.

As we can see, DevOps is a mix of software development, operations and services. It is a fusion of these disciplines to stress cohesion, collaboration, and communication between the conventionally distinct development and IT operations teams [15]. DevOps practices yield remarkable results for IT teams and organization [38]. DevOps practices are made possible by automation, both because it cuts out time-consuming manual work and eliminates human errors. In brief, Development and Operations work closely together under the common term that we call DevOps.

2.2 The rise of Agile and DevOps

Waterfall is a well-known traditional software development methodology, which used to be very popular among many organizations. Although this methodology is very simple and easy to understand and use, it also has many disadvantages. However, when Waterfall is dead, agile comes in and fills some of the gaps [58]. One major advantage of Agile is, it speeds up the delivery rate of the products and solves the problem of lengthy releases as well [32]. It helps product owners to define sprint backlogs, and development teams to prioritize work. Furthermore, it gives business or client the ability to say what works and what doesn't and see that feedback loop quickly. That said, a development team can rearrange tasks based on the bottleneck or business priority, which brings more flexibility. Usually, a typical application lifecycle consists of three major parts:

- Development

- Testing
- Operations

Typically, in Waterfall, these three areas are kept separate and are run by different groups. Despite agile solving the development and testing issues which were present predominantly in the Waterfall method, it does not include the operations [32].

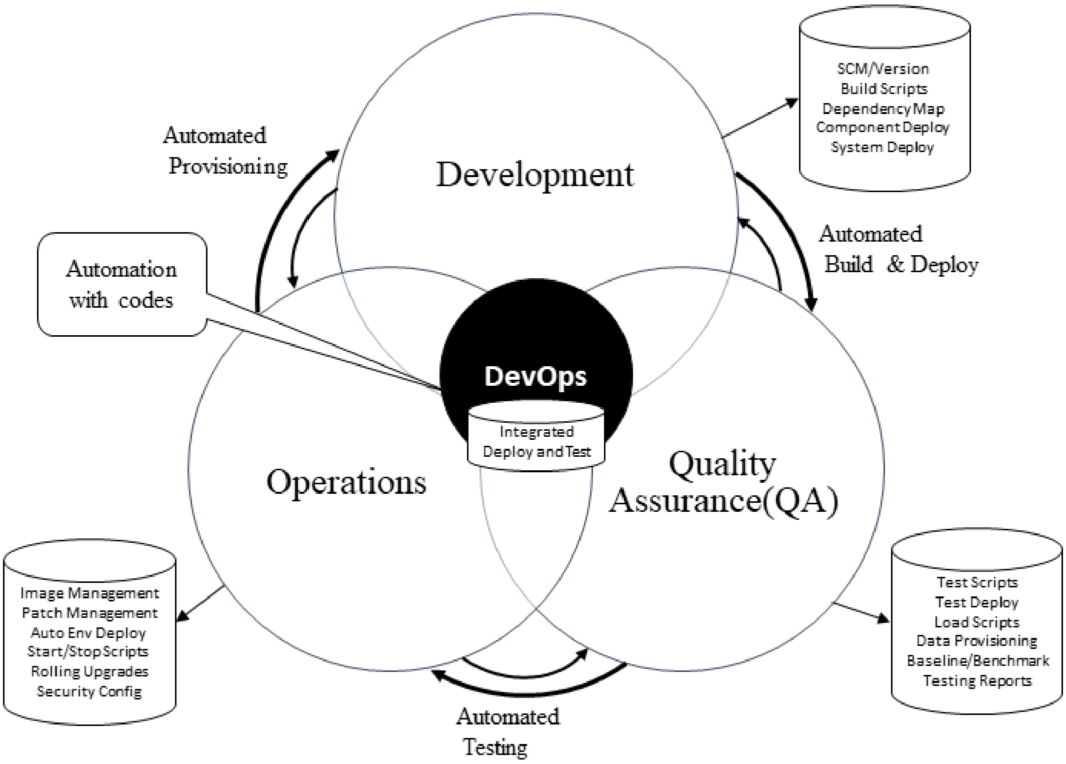


Figure 3: General overview of DevOps in the software development lifecycle [17]

One factor which has led DevOps in priority is- it brings these three groups together. To illustrate the ongoing discussion, the overall concept of the DevOps practice and how it fits into the agile methodology is presented in figure 3. Although Agile and DevOps are not the same but are typically closely associated. It allows the same group to perform all the functions. However, one justification that could be given for this is- Agile is a development methodology, and DevOps is a culture. This implies more like thinking about an application lifecycle. As a consequence, not the entire development can move quickly but the entire release can. The DevOps methodology empowers a team that closely define develop and release. As we see it, DevOps is a huge culture change not only for developers but also for the whole organization. Many tools have emerged that allow application development teams to work more quickly and efficiently than before. As a matter of fact, the adoption of a new set of tools is simple compared to changing the whole organizational culture.

2.3 The key areas of DevOps

DevOps involves automating the process of software delivery and infrastructure changes. The largest problem in most areas is lack of automation. Without mature automation in place, true DevOps culture will struggle. Achieving the velocity that needed will be hard. Two major sides of DevOps can be identified in the following characteristics:

1. Operation Centric:
   - Manage inventory of servers automatically- Provisioned, configured automatically
   - Monitoring analysis of operations
2. Developer Centric:
   - Continuous Deployment
   - Push code to production through the pipeline

To elaborate, in Dev section of DevOps, the main activities that took place are, build and release, run test cases and much more. On the other hand, the Ops section performs the activities such as server orchestration, provisioning, automation of almost everything that comes in the way. The main principle of any DevOps team is to automate everything from infrastructure provisioning to software testing and deployments. It is the standard way of performing operational activities for businesses. Also, the team is responsible for writing configuration management codes or scripts to make the deployment infrastructure to the desired state instead of configuring the software and hardware manually. The manual activities for configuration management, for example, application configuration, hardware specification, OS specification, Web servers, etc. is being gradually replaced by the implementation of DevOps. So, if the pure DevOps practice is in place, server provisioning, scaling, application testing and deployment can be automated [40].

Security has drawn a remarkable attention in the DevOps world. To enable security into DevOps practice, this has led to the rise of a new field called DevSecOps [1]. Prudent use of security automation into DevOps culture is to allow the teams to maintain both security and speed during the application development phase. In a nutshell, DevOps combines the needs and wants of multidisciplinary teams, and it brings many capabilities such as continuous planning, continuous integration and testing, continuous development, continuous infrastructure monitoring and optimization and so on. On top of it, DevOps requires both dev and ops skills as well as knowledge.

2.4 Test automation in DevOps ecosystem

DevOps involves automating the process of software delivery and infrastructure changes. Many organizations struggle managing workflows during continuous integration (CI) and continuous delivery (CD) due to the hands-off between development and operations stages [9]. Achieving automation across process flows is not an easy task. In addition, the power of automation in the DevOps lifecycle is huge. In DevOps context, automation is considered as a key to effective collaboration and integration between deployment and operations. As a matter of fact, DevOps community is also active in this case and they are consistently pushing new approaches, tools and open-source artifacts to implement such automated processes [50]. In addition to this, test automation can be defined as a set of assumptions, concepts and tools that provide support for automated software testing by adopting pre-recorded and predefined actions [23].

By empowering the integration technologies, we can bring the tools together as used by different stakeholders. In order to implement DevOps successfully, integrating participating tools is vital to automate process flow. Test automation is just the use of special software or tools to control the execution of tests. Tooling is required to implement end-to-end automation deployment processes. It should not be forgotten that time matters in agile or DevOps culture. Anything that we can do quicker will help the process succeed. One major fact in this context is the practice of configuration management automation to meet increasing infrastructure demands. In this thesis project, the research centered on how we can perform all our testings quickly for available infrastructures.

2.5 Automation in testing

In any software development process, one major part is devoted to running test cases. Typically, automated testing refers to a process of automating the execution of test cases. Before starting the discussion on test automation design, it is essential to define some of the most common terms related to the topic. Regarding continuous testing, there is a slight distinction that often needs to be considered when it comes to automation. Along similar lines, two types of automation underlying in the field of testing particularly in continuous testing [26].

- Automated testing
- Test automation

Although these terms seem to represent the same thing, at some point, actually it has underlying slightly different meanings. A closer look at the terms indicates that automated testing actually is the act of conducting specific tests via automation [26]. By contrast, test automation generally specifies automating the process of tracking and managing different tests [26]. Then again, the common goal for test automation is increasing the speed of test execution and also to increase the test coverage. With automation, including automated testing, we can promote work far more quickly through the pipeline. Also, it brings confidence, which ensures that systems are all working as they should be. Some advantages that test automation brings for us are:

- Saves a lot of time in the test process
- Helps to increase test coverage
- Allows to perform the unattended execution
- Enables parallel execution
- Supports execution of repeated test cases
- Ensure more accuracy by reducing human-generated errors
- Improves quality

In a typical scenario, there are fundamental reasons tells us what type of test cases to automate. Based on the available information, the following test cases can be automated [28].

- High Risk- business critical test cases
- Test cases that are repeatedly executed
- Test cases that are very tedious and difficult to perform manually
- Test cases which are time-consuming

2.6 DevOps tools and technological solutions

The recent trends, particularly towards DevOps, tell us the number of new technologies that are being released into the market is growing remarkably [47]. Some of the commonly used DevOps tools and related use cases are given in table 1.

Servers Provisioning Technologies	**Configuration/Deployment Management Tools**	**Continuous Integration**	**Infrastructure Provisioning**
AWS	Ansible	Jenkins	Terraform
OpenStack	Chef	Hudson	
VMware	Puppet	Bamboo	
Cloud front	SaltStack	Travis CI	
Microsoft Azure	uDeploy		
Google Cloud			
Digital ocean			
Artifactory Management Tools	**Source Code Version Management Tools**	**Build Tools**	**Infrastructure Monitoring Tools**
Nexus	Bit Bucket	Maven	Nagios
Artifactory	GitHub	Ant	Prometheus
JFrog	Git lab	Gulp	
	Subversion	Gradle	
	Perforce		
	CVS		

Table 1: List of commonly used DevOps tools

This is, however, the most challenging part of knowing how a DevOps oriented team can accomplish the things discussed earlier. In the software development process, the DevOps tools and technologies are predominantly applicable to these particular use cases:

- Machine provisioning
- Configuration/Deployment management
- Continuous Integration
- Artifactory management
- Source code version management
- Build systems

In the DevOps domain, automating different technologies is beyond limits and people constantly working with integrating numerous systems. There seems to be no compelling reason to argue that technologies like clouds have changed the expectations for development team exponentially. AWS, Microsoft Azure, Google Cloud, OpenStack, VMWare are the most popular cloud providers among many organizations. Especially in the malware testing process, cloud service has remarkable contribution because of virtualization technologies in available computing infrastructure services.

This section lies at the heart of the discussion of how these above mention tools and technology work together. With this in mind, in the development process, it is a compulsory practice to maintain the application's source code using version control systems. Github, Git lab and Bitbucket are some of the most commonly used tools for the source code version management segment. CI tools such as Jenkins, Hudson, and Bamboo are mainly used for automating code test, build and deploy. Using these tools, we can get the latest code automatically from the version control systems (VCS). Moreover, CI tools also have the extended capability of automating the infrastructure provisioning and destroying with the help of configuration management tools.

Some of the most popular tools in DevOps toolchain, for instance, Puppet, Chef, Ansible and SaltStack provides different paths to achieve a common goal of managing large-scale server infrastructure and deploy the code to different environments efficiently. By utilizing these tools, we can actually code our infrastructure to instruct how it should look and behave. Furthermore, to store the executable artifacts, Nexus and JFrog are widely used across many organizations. Moreover, containerization technology, for example, Docker is a big name in the DevOps ecosystem, which allows running distributed application in a single virtual machine without launching an entire VM for each app [45].

One good thing is that these tools require very minimal input from developers and sysadmins in order to manage those infrastructures. They are designed to reduce the complexity of configuring distributed infrastructure resources. From the point of view, these toolsets can be seen as the operating systems of the future. Using the right tools, be it for testing or from an application's development to the production environment, paves the way to get faster and better outcomes. Furthermore, there are many tools and plugins with many more capabilities are introduced to the market every single day.

2.7 Automation framework Ansible

Application developers started to define their environment expectation which translates well into configuration management directives. Ansible is one of the most common DevOps tools used these days for automating configuration management and deployment. One good side of using Ansible is getting the benefits of both configuration management and deployment in a single tool, which makes the operation tasks much simpler [18]. The directives in Ansible are expressed in a way that both developers and operators can understand.

In addition, the Ansible engine has minimal installation requirements that basically requires Python with a few additional libraries. On the other hand, agent software is not required on the host that will be managed. The action Ansible takes on target hosts is called tasks which is a descriptive bit of YAML code written by the developer in order to complete the desired action on remote machines. In this project, this tool has significant applicability throughout the testing environment management process.

2.8 Virtualization in Test Automation

Virtualization is not a new technology. The available information indicates that the concept was developed back in the early 1970s by an IBM programmer Jim Rymarczyk, later it severed as inspiration for VMware [7]. This is the technology which allows running multiple machines utilizing a single hardware resource [24]. However, in the domain of cloud computing, it plays a major role as it provides virtual storage and computing services. VirtualBox and VMware are popular virtualization platforms which are able to spawn one or multiple parallel machines. Isolation of applications through virtualization increases security compared to the traditional bare metal deployment model [1]. The most significant part of adopting virtualization in the test automation context is the achieving of high uptime of mission-critical systems. It provides the ability to delete, recovering and re-provisioning the infected machines easily. Nevertheless, the cloud provides an environment, rich with automation opportunities.

3 Overview of the Target Environments

Since vendor's client testing mainly involves installation, manipulation and uninstallation of security products, different virtualization solutions are commonly used for running machines in order to run the tests. Throughout this thesis, the terms 'Infrastructure' and 'Test Environment' were used interchangeably by the practice of the department where this study was conducted. To give an illustration, infrastructure is often refers to whole physical machines or hand managed virtual machines provisioned from limited capacity [49]. Similarly, a test environment refers to a setup of software and hardware [11].

The main goal of this section is to provide an overall description of the environments which were used for the test execution. It also highlights some issues that were found during the analysis of the systems. On top of that, a general overview of the actual implementation plan for the test automation is included in the description. Due to a non-disclosure agreement, a detailed description of some part of the internal systems was outside the scope of this thesis. At this point, this chapter describes the minimal outline of the different test environments and presents the most prominent interaction points of this project. However, the actual architecture and explanation of implementation the method related to this solution will follow in chapter 5.

3.1 Pluggable test environments

The purpose of OneTA solution is to initially cover pluggable supports for five different test environments. There are mainly four main internal environments, and they are fully owned and maintained by this thesis commissioning organization. Each of them has a specific purpose of use. The terms "Extensible" and "Pluggable" both are closely related but underlay slightly different approaches. "Pluggable supports" here, represents the ability to remove the environment or substituting according to the needs, whereas extensible generally refers using the application or environment from its base. In short, the pluggable approach provides the ability of just dropping any of the environments and keep the other environments still usable. The proposed solution includes an in-house private cloud infrastructure, a sandbox system, pre-configured physical or bare metal machines and an isolated OpenStack cloud-based real malware test environment as part of the internal infrastructure. On the other hand, one other environment is AWS (-EC2). Table 2 summarizes the overall use cases of the target environments which were primarily used and implemented for running the tests.

Target Environments	Test Use Case
Bare-metal	AV product testing against physical hardware-based test machine
DVMPS	AV product's performance analysis against windows environment
AWS-EC2	AV product's performance analysis both windows and Linux networked public cloud environment
Sandbox solution	Execution of unknown files and URLs (including malware with possibility of behavioral metadata extraction)
Red Cloud	Execution of any kinds of unknown files (including malware)

Table 2: List of target environments for conducting the tests

3.1.1 Bare-metal environment

It is a common finding that there is a performance variation between the physical machine and the virtual machine even with the same number of cores [22]. The term "Bare-metal" is used nowadays to distinguish the physical machine from modern forms of virtualization. As a matter of fact, it should not be denied that not everything behaves the same in virtual and physical machines. One important aspect in this case is to note that it is often not possible to get both the systems running on the exact same HW and environment, thus, consumer level results might not be same in both scenarios. Having said that, hidden contention for physical resources may impact performance differently in different workload configurations [53]. As for the causes, a significant variance is often noticed in a system throughput.

By running multiple VMs in a shared physical machine, we can enable high utilization of hardware resources. In the virtualization technology, hardware utilization is achieved by using the technology called hypervisor, which provides access to the physical machine and allows sharing of CPU resources. In the testing process, it is a common approach to conduct the anti-malware product's performance testing against "single-tenant physical machine". For the sake of validation, performing the same tests in virtual and physical machines is essential. One of the most prominent use cases in this context was to validate if problems reproduce both in physical and virtual machines. Additionally, bare-metal tests are often run against consumer grade hardware to understand the actual customer experience. The power or the performance optimization methods may create a big difference on various workloads. It provides more visibility what the end user would actually experience when it comes to performance of the vendor's security solution. It gives the QA team more understanding about what to expect from the actual setup. In order to get the exact views of the test cases, OneTA also includes the support for running the anti-malware product test against the physical machine. It allows executing commands on a remote physical machine which is already provisioned. In this solution, the connection to the physical machine was implemented using SSH. By including Bare-metal in the test scopes, the result can be compared with test running at the same time in the virtual machine or analyze actual consumer HW grade scenario.

3.1.2 Dynamic Virtual Machine Provisioning Systems (DVMPS)

One major part of this TA process was to orchestrate the test execution process in one of the legacy on-premise virtualization platforms called DVMPS. In testing operations, this kernel-based virtualization platform or KVM is mainly used for performing the test in

Windows environments. In this virtual machine provisioning system, each machine has private virtualized hardware such as a network card, disk, graphics adapter, etc. and is accessible via the internal network. It allows few options for creating the disk image, for instance, installing the guest OS from scratch or converting an existing guest image to KVM qcow2 format. The QCOW image format is one of the disk image formats supported by the QEMU processor emulator [46]. KVM specifically eases Linux to turn into a hypervisor and therefore, allow a host machine to run multiple isolated virtual environments [51]. Virtual environments here referred to guests or virtual machines (VMs).

However, while dealing with windows environment, there could be many options, but not all of them provide a way to make it easy to test software and then roll back to a clean state, and on top of that there is a question of licensing requirements. Because all Windows computers, be it a real physical PC or a virtual machine have a unique ID. Using the same ISO image, clean installation on a new VM, the machine gets totally new ID and signature. This arises the issue of licensing requirements. There are different implementation approaches adopted by the developers of this platform to eliminate the dependency when it comes to Windows OS. This in-house virtualization solution allows reusability of windows images by creating “Machine Snapshot”. A guest operating system is created on the host server, and similarly, new machines are provisioned on a whim by accessing the environment through the internal network.

In this thesis scope, OneTA needed to fulfil the requirements to automate the provisioning of VMs from available templates and run the test by integrating the remote execution tool as a part of the OneTA library. DVMPS consists of a wide range of windows templates. Machines here configured from these templates that enable implementing the test mainly in Windows virtual environment using specific test execution methods.

Virtual machines in DVMPS have a short lifespan and intend to use and complete the test in a maximum of two hours. The platform provides simple GUI in which machines are manually created and provisioned by following few steps. Figure 4 depicts the manual steps required for creating a test environment in DVMPS platform.

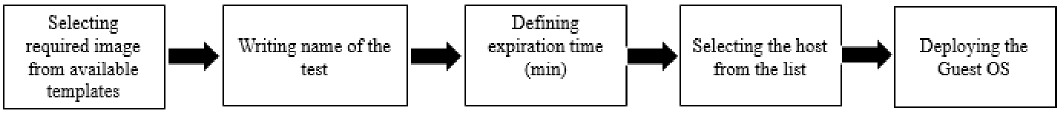

Selecting required image from available templates
Writing name of the test
Defining expiration time (min)
Selecting the host from the list
Deploying the Guest OS

Figure 4: Manual steps for creating VM in DVMPS environment

Once the selected VM is provisioned, the new machine is available in the currently active machine list and accessible via VNC software such as TightVNC. Each test VMs in DVMPS includes the following information:

- Name of the test
- Expire time
- Template name
- Image ID
- IP address
- VNC, e.g. 10.133.32.23:5908

Virtual machines in DVMPS require a particular remote execution tool called "FSExec" in order to access and execute commands remotely on Windows systems, which was developed by this thesis commissioning organization. Furthermore, FSExec is one of the major components of the OneTA and slightly modified version of the originally developed version. A client software installation for FSExec is required in every VMs in order to perform remote test execution. Additionally, all the windows machines in DVMPS are preconfigured with FSExec client installation during the creation of the image. In this project, the creation of new machines in DVMPS was fully automated through Python. OneTA includes FSExec as a compulsory unit and is only used for executing remote commands in DVMPS environment.

3.1.3 AWS EC-2

Amazon Elastic Compute Cloud (Amazon-EC2) is one of the most popular web-based cloud computing services in enterprise level that provides secure and resizable computing ability [27]. The idea of having public cloud infrastructure is the ability to rent virtual computers on which to run computer applications without concerning the hardware. Amazon Web Services (AWS) is a commonly used public cloud computing platform in this thesis commissioning organization for various services.

Complete orchestration of anti-malware product tests against some use cases in AWS environment was one of the objectives of this automation solution. The pluggable support for this publicly available infrastructure was implemented to create a virtual machine and run various kinds of tests, for instance, load testing, acceptance testing, performance testing, etc. against vendor's anti-malware software products. AWS is very effective in each of these scenarios and phases. Software configuration, for example, the operating system in AWS is prepared upon a template called Amazon Machine Image (AMI). In addition, virtual machines in AWS-EC2 are called instance, and furthermore, AWS provides many publicly available AMIs containing software configuration, which provides complete control of computing resources. Instant remote access to the machine can be established by using SSH.

In DevOps culture, it is not worthy to manually set up the EC2 instances, therefore, automating all the EC2 builds by provisioning only the resources needed for the duration of development phases or test runs was a major concern in this case. In response to EC2 instance creation, AWS provides the ability to set up a development and test infrastructure within a minute [37]. AWS "Access Key ID" and "Secret Access Key" are required to create an end to end secure connection. In the Ops part, there are still some manual tasks involved, for instance, in creating a security group- that needs to be done while preparing the AWS account for a particular user group before automating the provisioning of the resources.

While AWS focuses on more efficient lifecycle on the "Scriptable infrastructure", OneTA enables solutions that provide only a subset of the functionality for infrastructure provisioning by using compatible DevOps tools. In OneTA approach, the AWS configuration management was implemented by using the popular configuration management tool Ansible.

3.1.4 Sandbox solution

To enable more scope in the project, OneTA includes a cloud-based sandbox environment as a part of the pluggable approach which allows the execution of unknown objects such as files or URLs in a safe environment and generates an in-depth report about their behaviour. Typically, a sandbox referred to an isolated computing environment in which a file or an unknown object can be executed without affecting the application in which it runs [39]. Cuckoo and Malware Jail are two popular open source sandboxing solutions, which can be referred to in this case [12].

As detecting and removing malware artifacts in endpoint protection is not enough these days, it is vitally important to understand how they operate in order to understand the actual context. The sandboxing concept widely applied in malware analysis to run an unknown and untrusted application relying on signature-based scanning to detect and block malicious activity [43]. In a typical scenario, an attacker only needs to bypass behavioural analysis components in order to infect the system [41].

This sandboxing solution was developed with years of industry experience and maintained by this thesis commissioning organization. While this sandboxing solution focuses on the "Deep Analysis" enabled by the automated malware analysis system, OneTA utilizes the technology to provide smaller solutions to some use cases where unknown files or URLs can be tested in continuous integration practice. The main intent of this inclusion was to integrate the vendor's easily deployable extensive threat intelligence automation solution. It is a cloud-based service and uses a Black box approach in the automated analysis process. The solution consists of different components of the malware analysis technology on a dedicated system that provides a thorough analysis of any given files and URLs. All the communication between different components takes place within an encrypted network, and due to its technical necessity, the network is labeled as a part of the Red network.

The web-based service allows throwing any suspicious files or URLs into this environment, and therefore, it generates a set of information outlining the behaviour of the file, which can be retrieved with the related task ID once ready. In the testing process, the submission of files and URLs was implemented using REST API. A hooking API allows making a request of submission of objects in the sandboxed environment. However, a common approach in the headless testing process is storing the checksums (hashes) of files and schedule a detonation for a file with a given SHA1 or SHA256 value.

3.1.5 Isolated malware test environment: Red Cloud

Red Cloud is an OpenStack-based virtualization environment that runs in an isolated malware testing network called Red. OpenStack offers a free, open-source and IaaS based software platform for cloud computing. More details on this topic can be found in [29]. Since malware is very disruptive in nature, considerable attention must be paid when executing them for the test purposes. This cloud computing environment was developed and maintained by the thesis commissioning organization with the goal of executing real malware. The environment allows execution of any malicious file in an isolated network without compromising the security of the safe network.

However, the network or the environment is labeled as “Red” due to the nature of handling and executing live malware. The Red environment is intended to supply the following purposes:

- Replicate the "real world" malware tests performed by the third-party organizations (VB100, AV-Test etc.)
- Test and compare the detection capabilities of vendor’s current and next generation AV products
- Test the latest (alpha) AV engines with real malware

Red Cloud is a technical necessity for highly malicious use cases, and it requires a technically separate deployment. That is to say; virtual machines are completely isolated by default in this cloud environment. Furthermore, test machines in the red environment are either virtual machines in OpenStack, or physical machines plugged in directly to the Red network. The overall infrastructure of the “Red Cloud” spans several network environments in order to support safe handling of live malware.

Red Cloud includes the malware isolation capabilities of OneTA and provides a broader scope of the solution as stated in the thesis objectives. In addition to that, OneTA utilizes Red Test Automation (RedTA) as a pluggable component in user-level context, since the overall infrastructure was fully developed by this thesis conducting organization.

However, multiple environments are used within the organization to maintain security among different components of the network and they are labeled with different names based on the use cases. The network fragmentation of the overall structure is divided into three levels and they support different purposes. Table 3 summarizes this project's related environments and their main purposes.

Environments Name	**Purposes**
Green or Blue environment	• Used as a common test environment • Facilitates functional and non-functional test cases • Does not allow any malware samples to store and execute
Orange environment	• Allows the storage and handling of raw/unencrypted malware samples • Allows static analysis of malware samples, but the execution is not permitted
Red environment	• Allows similar functionalities as Orange environment • Scanning, storage and execution of any type of malware samples (e.g. EICAR) are allowed

Table 3: Purposes and labeling of different environments inside the organization

The primary intent of this project was to communicate with the Red environment by understanding the overall system architecture and requirements, therefore, providing an interaction point between different components of the Red Cloud service to fulfil the Red TA use case. Moreover, the test system implemented for the Red TA use cases required to integrate the continuous integration workflows of the organization. That is to say, the automation server for hosting the Red tests is nested in the Green environment which provides the interaction points for OneTA. All other interaction points, for instance, the Orange environment for malware storage and test execution in the Red Cloud initiated from this automation server machine which is actually a Jenkin specific special slave machine or OneTA controller machine.

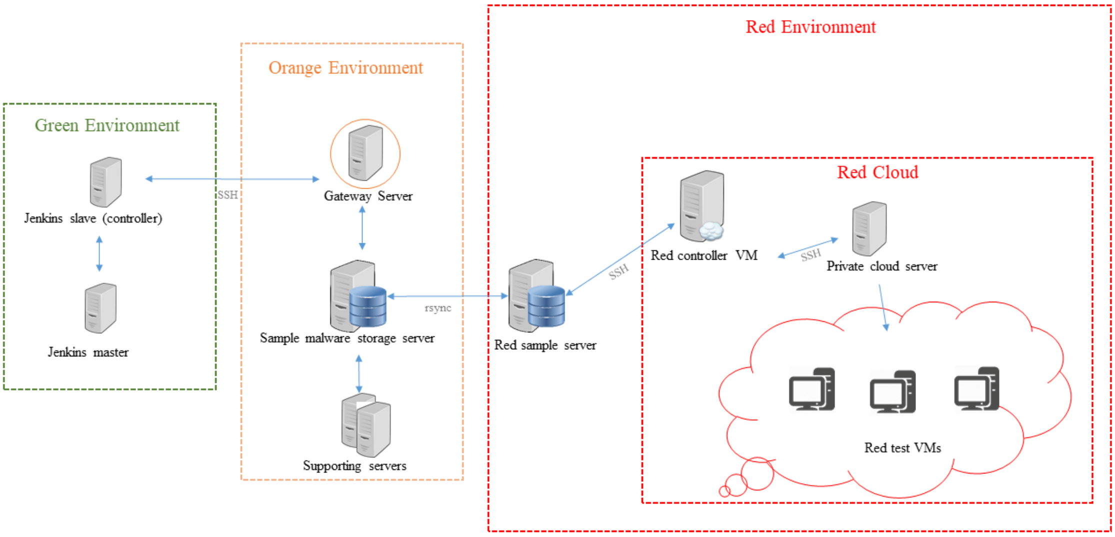


Figure 5: Overview of the Red cloud architecture and purposes of labeling

Since Red cloud is highly secure and isolated due to technical essentiality, there is no direct connection between the Green and Red environment. Figure 5 depicts the overall scenario that illustrates how Red cloud interacts with the Red platform. Required tests artifacts, for instance, test cases from the Green environment to the Red environment are transferred using a special “Gateway Server” which is placed in the Orange environment. Only specific Jenkins slave has access to this Gateway channel in order to transfer files and initiate the execution of live malware in the Red environment. The actual process in the Red Cloud is highly technical in nature, and it follows certain rules to keep the other environments safe. Only selected slaves in the green environment have firewall access to machine in the Orange environment.

4 Defining the Test Related Requirements and Problems

While dealing with different test environments, there usually involves some challenges and that can be seen internally and externally. For example, the compatibility of different tools and approaches for different environments is often uncertain. Similarly, different teams inside the organization have different ways of achieving the same goal. The objective of OneTA orchestration solution was to address a solution to those commonly identified problems so that it can minimize the effort, especially for the test managers in various test scenarios. With this in mind, the final selection of tools and approaches were made based on the consultation of senior engineers who have already worked on those systems and have better understanding regarding commonly known issues.

The implementation of this test orchestration solution required clear understanding on target infrastructures that were discussed in the previous chapter and development of corresponding execution methods using the Python programming language to achieve a common goal in various abstractions. In this project, the operational requirements were derived from organization's choices and industry standard practices. The challenge was to find a way to make testing simple, effective, and automated in number of pluggable environments. This section tries to define some of the common issues, requirements and considerations while reviewing the target environments during the test automation solution process.

4.1 Specifying the functionality

The main functionality of this solution was to allow test execution for desired use cases utilizing DevOps practice on top so that it can support those target environments under single deployment. OneTA supports multiple synced execution types, multiple provisioners to set up the machine, automatic SSH, API request, creating secure tunnels into the test environment, and more. These can be configured using JSON files as a common entry point. In the test process, OneTA consists of different test phases where various kinds of tests take place, including load testing, performance testing, detection capabilities testing etc. in a single setup. In the process of test environment automation, the tasks mainly include: automating configuration, refreshing test data and deploying the software to the test environment. The execution of automated tests should be followed by after that.

4.2 Infra instability and inconsistency

The backend infrastructures mainly consist of web servers, application servers, databases, task queues, etc. which run in a distributed set of computing resources and communicate through different protocols. The primary need was to deploy and maintain virtual machine instances in services such as Infrastructure-as-a-Service (IaaS). One of the major issues in this kind of infrastructure is the availability of enough computing resources during the test execution process. In a typical scenario, if the service is unable to provide enough resources, for example, memory or disk during the process, it may cause data loss. It is not unusual to see that service is broken during the test execution process. Due to the unavailability of the resources may result in test failure. There are a considerable amount of network issues that were observed earlier in one of the legacy infrastructures of the target environments. This is, however, found that the network is often unreliable, and the connection is being cut in middle of the test run.

4.3 Future consideration for a widespread range of cloud platforms

Test environments can be quickly and easily built across a wide variety of cloud platforms such as Amazon EC2, Google Cloud, Microsoft Azure, DigitalOcean, CloudStack, OpenStack and many more. Although these platforms fulfil similar demands, the best option may not always involve just in one cloud provider. It is a common fact, decision-makers in the organization select some platforms and teams must go with the change, or some project or test may require newer or older software/hardware systems. These changes in test environment management are obvious and different features in each platform play a crucial role in this case. Often, Legacy IT systems are not prepared for the change. In this scenario, OneTA adopted a pluggable approach so that new systems, meanwhile, are far more flexible and easier to adopt in the future on an as-needed basis. That means OneTA should have the capability to support different types of platforms in future with the similar configuration.

4.4 Security considerations related to handling real malware

OneTA intent to provide a solution which is minimum viable in the security context. For example, in the development phase, it was considered how we can set up tools and processes such as version control, collaboration environments, and automated build processes securely and durably. In the testing phase, the focus was on how to set up test environments in an automated fashion, and how to run various types of tests including a real malware.

In this scenario, especially, while handling real malware; a considerable amount of caution should be taken to keep the network safe. Since the creation of test suits and storing the malware sample was outside the scope of the OneTA solution, the main activities involve enabling the service required to initiate the test flow to the red environment from the green environment. Usually, the organization facilitates special training for those who are directly involved in administering the Red environment. OneTA implements the logic and follows the procedure to run the test programmatically so that it fulfils at least the minimum security considerations while dealing with an isolated malware test environment.

4.5 Main requirements for OneTA solution

As OneTA seeks to solve some in-house test related issues, all in all, it primarily targeted to meet the following requirements:

- OneTA can be used as a common solution for various test related activities in multiple environments.
- The solution must allow continuous integration practice considering Jenkins as a base.
- Primarily, it should be able to provision new test machines in AWS and DVMPS environment.
- The solution must be able to integrate Red environment with proper security consideration.
- All the tools and other related dependencies should be covered in one space so that it can be used as an independent tool.
- The network connection, test coverage and test execution method in the target environments must be automated and stable.

5 Design and Implementation of the Orchestration Solution

This chapter covers the broad view of the actual technical work of the OneTA solution. It describes definitions of the logic and control flow, explanation of the tools used for the test execution resulting from OneTA development phase. It covers the functional requirements and specifically, focuses on the actual server-side test automation implementation such as how they were executed and how the results were processed in the main workflow. The actual development phase concentrated to implement a Python-based workable test automation library for target environments, thus integrated with the existing systems and continuous integration flows.

5.1 Logical architecture and main workflow of OneTA solution

To give a visual illustration, the overall test process and the principal of OneTA solution are described in figure 6.

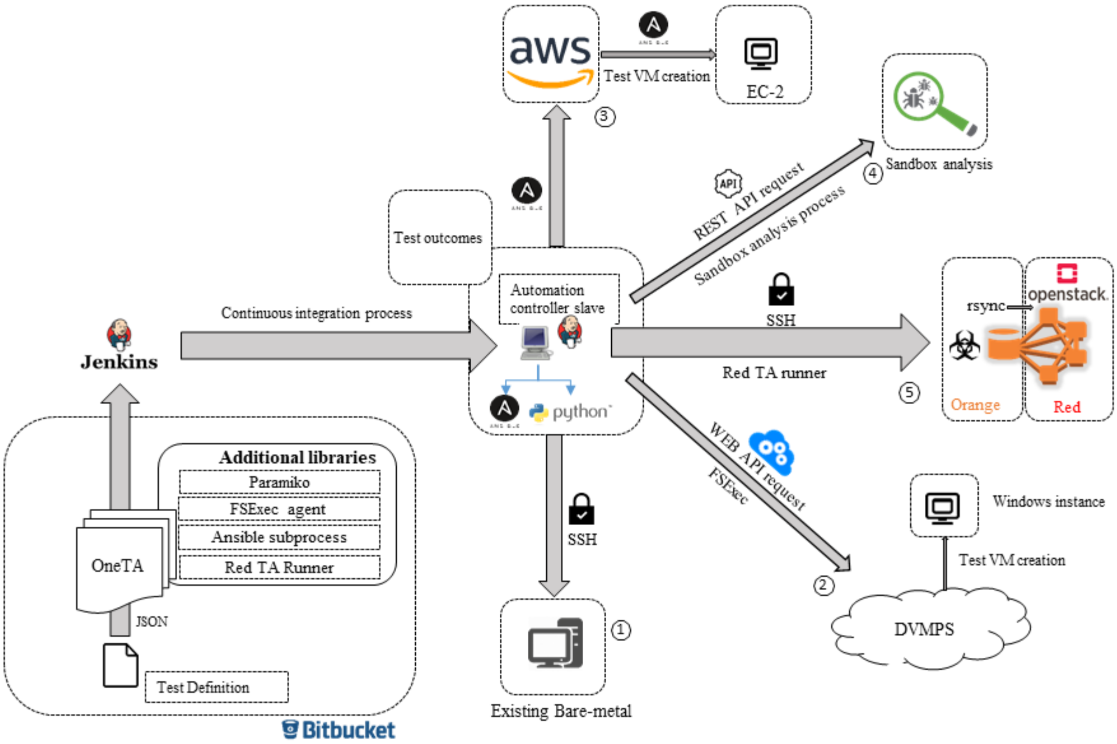


Figure 6: Actual system design and principal of workflow of the OneTA solution

Several test related services OneTA covers, consisting of physical and virtual infrastructures. The first part of the test process, test planning and test description, is vital because it defines objectives of the testing and specification of the test activities. Then again, the end-to-end full solution includes a total of five different environments for test execution, including a sandbox solution and an isolated malware execution environment.

The test specific solution throughly follows the DevOps approach in the test execution method similar to other test jobs available in the organization. The actual workflow consists of a few major areas of DevOps toolchain such as source control management (SCM) tool Git, continuous integration tool Jenkins and configuration management tool Ansible. On top of that, there were other in-house remote command execution tools, for instance, FSExec used in the main workflow. In addition, OneTA utilizes Red TA runner for executing real malware on the Red cloud. Automation scripts are executed during this phase, and they require the input of test data before being set to run.

The main workflow in the current solution maintains a specific sequence during the test process. The first test process starts from the Bare-metal environment, and final test process is Red test execution. The sequence of the test processes is also numbered in figure 6. However, OneTA involves the creation of test VM only in the AWS and DVMPS environments during the test flow. Moreover, it includes some networking protocols that interact together in the scope of an automatic network connection. While the connection and configuration are established in the target environments, it initiates the test execution. Test outcomes or any error messages during the test flow are visible in the test controller server machine's console output, and test results are available in the Jenkins artifactory.

5.2 Fragmentation of tools and required resources

OneTA uses both in-house and publicly available tools and resources to cover the overall testing process. The related testing tools do not perform the actual testing by themselves. Indeed, a set of preconfigured tools nested in its service lifecycle loop so that they can facilitate all the planned test cases. Table 4 summarizes the core components and their role in the test orchestration process of the OneTA solution. Each of these components are essential elements of this solution to obtain the full outcome.

Name of the component	**Role in the testing process**
Git Server (Bitbucket)	• Used for source control management • Used for storing OneTA core project files containing source code • Also includes FSExec agent scripts for remote execution in DVMPS environment
Jenkins	• Used for preparing “Automation Server” • Allows maintaining the continuous integration and continuous delivery practice • The controller Jenkin slave machine pulls the project source codes automatically
Automation server	• One of the special Jenkins slave machines, which can initiate the test in the Red Cloud. • Used for accessing the in-house networking tools and also eliminates the manual activates that mostly required for the Red use cases
TA Runner	• Special agent used for establishing the communication into the Red environment.

	• It has selective test running capabilities to run the test in Red Cloud
Sandbox REST API	• Used for making API calls to automate the task of analyzing any malicious file or URL • Also responsible for retrieving the general behavioural information of the file
FSExec agent	• FSExec is responsible for running remote commands and also uploading and downloading files to and from the VM in DVMPS See section 5.5 for more details.
FSExec client	• FSExec client is installed in every windows VMs in DVMPS environment. • Must require tool for executing a remote command in the VM
SSH	• Used to log into a remote machine and execute commands securely • Public key authentication was implemented for passwordless login to the remote host.
Ansible	• Responsible for configuration management, application deployment, task automation in AWS environment.

JSON	• Used by main Python script to read the test definitions in human-readable format • Default JSON configuration files provide the guideline for test related configuration
YAML	• Used as a configuration file for AWS environment provisioning • YAML only meets the requirements for AWS use cases
Payload	Payload contents: • Test related scripts, libraries, tools etc. • Remote shell or bash commands

Table 4: List of OneTA components and roles in the test process

5.3 Backend and client-side infrastructures

The solution this thesis presents covers three main functional areas and they are based on these following domains:

- End-to-end Testlab infra
- Virtual and Bare-metal environments
- Client-side test automation

The Testlab infrastructure is fully developed and typically maintained by the system engineering teams of the organization. The backend infrastructure is mainly involved in delivering the artifacts to and from the test environments after the test process. In the test execution process, OneTA uses specific test-runner scripts (Python scripts) in order to interact with internal virtualization solutions. The test-runner is the core component in terms of running the actual tests. Similarly, the test controller machine or automation server subject to the high-level abstraction of different operations on the VM. The Testlab infra also spans to the Green, Orange and Red environments, was discussed in the previous chapter.

The client-side automation mainly executes the actual tests scripted by the system. There are mainly two kinds of test scenarios in the test process. Having said that, the tests can be outlined in terms of “test cases” and “test sets”. For instance, “test cases” commonly test a single feature of the anti-malware software. On the other hand, “test sets” are generally understood to mean a collection of test cases which has been applied to the same test session. Furthermore, OneTA exposes all necessary endpoints of the test scenarios and executes them to the corresponding client-side infrastructures in a single setup.

5.4 Test machine creation in DVMPS environment

One of the major use cases of this orchestration solution was to automate the test process in the DVMPS environment. This is, however, a poorly maintained platform but provides usefulness in many use cases. In addition to that, several technology templates provide the service, but there was no comprehensive orchestration solution available for it. As mentioned earlier, DVMPS environment intends to use for running the AV product test against Windows-based operating systems. The main purpose of including DVMPS in OneTA is to provide the orchestration solution by performing the exactly needed activities in the target network in order to create a new machine from available preconfigured machine images and executes the commands for running the test using in-house remote execution tool.

DVMPS is hosted in the in-house server and accessibly via VPN. Since OneTA automation server machine has the full access to the DVMPS host, and both are connected to the same VPN, Web API request was considered as the basis for the DVMPS orchestration solution. OneTA is scripted to create a new request in DVMPS host using Python **`request.post()`** method with inclusion of the DVMPS configuration properties such as base image name, test name, machine expiration time and the allocated host name. When the machine gets created in the DVMPS environment, it returns the JSON response with machine's information, for instance, machine address, port number, machine name etc. To perform rest of the actions in the machine, OneTA includes FSExec in the loop to fulfil the test process. However, the FSExec agent only requires the machine address, typically the IP address of the newly created machine in order to perform the rest of the tests related activities inside the machine.

5.5 Inclusion of in-house remote execution tool "FSExec"

Both WinExec and PsExec allow launching interactive command-prompts on remote systems [48] [31]. The main purpose of these windows utilities is to execute a command-line process on a remote machine. Since test cases in the DVMPS environment require executing commands with administrative privileges, it is required to use a remote execution tool for making the target VM to run commands and download and upload test related files.

However, the additional problem is that, earlier the security research team in this thesis conducting organization noticed network issues with those utilities during the test run in the DVMPS environment. One of the drawbacks to adopting those utilities in the Testlab infra is that the connection is being cut or raises frequent network issues during the test execution process. This raises many questions while implementing end-to-end automation. The finding related to this use case tended to suggest that it is often due to vendor's own anti-malware products that terminate the connection. On the other hand, it is also required to have AV products installed in the target VM while testing some features of the product.

"FSExec"- the in-house remote execution tool is a solution for this issue which is also similar to PsExec and WinExec type approach with a more reliable solution. Additionally, it also provides the functionality to get and put files in the remote systems without relying on "Windows shares" and "Samba" utilities in the Jenkins controller machine. FSExec was used as an integral part of the OneTA solution in order to fulfil the test execution process in the DVMPS environment. The hard-coded solution of FSExec was included as a part of the OneTA library and only used for running the test in the DVMPS environment. OneTA included the functionality of FSExec using the Python subprocess module since it is usable via Command-line.

5.6 Python Paramiko for SSH connectivity

One of the use cases of this solution is to execute remote commands in a bare-metal machine which is already provisioned. In order to log into any bare-metal machine and execute commands, the client-server based SSH approach was adopted into this solution. Although there are other communication protocols used for remote communication, SSH is commonly used and provides a very secure encryption that protects the communication between the client and the server [56]. In this technique, the client machine is responsible for authenticating using a password or private key and checks the server's host key. The server machine is responsible for deciding which users, passwords, and keys to allow, and what kind of channels to allow [56]. Unlike SSL, SSH protocol does not require certificates signed by a certification authority, and the key exchange mechanism is fairly easy.

Paramiko is one of the popular Python modules for implementing SSHv2 protocol and available under GNU license. Although it depends on third-party C wrappers for low-level crypto, it is entirely written in Python and moreover, it is a pure Python interface around SSH networking concepts [54]. Moreover, it is also the low-level SSH client behind the high-level automation framework Ansible. To accomplish the tasks of accessing the bare-metal machine and controlling the command prompts, Python Paramiko was included in the solution. The end-to-end automatic network connection to the bare-metal machine was implemented on top of this native Python SSHv2 protocol for initiating the test using public key authentication.

5.7 REST API for the Sandbox solution

OneTA solution includes the vendor's own Sanbox solution in the test scope which includes scheduling files using REST API to analyze and later collects those results. The main task for this implementation is to submit an analysis through its web application using given <SHA1>, <SHA256> or <URL>.

Since the actual REST API for this in-house Sandbox solution was on on-going development during this project implementation period, a different approach was adopted in order to simulate the actual scenario to this solution so that it can work afterwards. Having said that, this was done by using API mocking, more specifically using Python mock objects. However, the initial plan was made using the conditions outlined in table 5.

Suggested API methods	**Usage**
POST /api/schedule	Schedule the analysis with given <SHA1>, <SHA256> or <URL>
GET /api/results/report/<TASK_ID>	Retrieves a JSON of the behavioral information
GET /api/results/report/<SHA1 or SHA256>	Retrieves a JSON with the latest known behavioural information
GET /api/results/artifacts/<TASK_ID>	Retrieves a zip with all the artifacts collected in a particular execution

Table 5: Sandbox REST API use cases

The API mocking was implemented through Python scripts and tested in the main workflow to simulate the actual scenario during the test process. Technically, a mock is a fake object that we can construct to look and act like real data [3]. This approach is useful when we need to make the request to the API endpoint, and returning outcome depends on the live server. The main goal of this mocking was to simulate the actual process of scheduling the analysis, then wait for some time and get a response of the behavioural information of the object in the JSON format. The mock data that was used in the simulation process was based on the assumption that the real data would also use. For the scope of the project, the main step was making a call to the actual API and taking note of the data that was returned.

5.8 Ansible for AWS configuration management

The OneTA solution includes Ansible to create, execute the test, terminate, start or stop an instance in AWS-EC2 in the test orchestration process. Other tools, for instance, Chef or Puppet could be options for this particular purpose, but configuration management tool Ansible was justified as a better DevOps tool especially for AWS instance provisioning. For pluggable environment support, it has modules (also known as “module library”) to create infrastructure, as well as modules to assert the configuration on that infrastructure. The reasoning behind this is to allow achieving higher level solutions and ensure a pluggable approach support across a large variety of platforms in the future. On top of that, it is highly scalable which means we can control ten machines, or we can control ten thousand machines in a single setup.

However, Ansible by default manages remote machines over SSH (Linux & Unix) or WinRM (Windows), and they already exist natively on those platforms. The main purpose of including Ansible into solution was to deploy VM with configuration assets to ensure automation of test workflows and test environment in AWS environment. This action was performed by using AWS specific Ansible “Playbooks” modules. In addition to that, Ansible’s Playbooks are written in simple YAML and used as a configuration, deployment, and orchestration language. The current solution includes a primary structure of a playbook to test the functionality of the OneTA solution in the target environment. In the simplest form, the Playbook can be run from the command-line of the machine where Ansible is installed in the following way:

```
$ ansible-playbook playbooks/sample.yml –i /ansible/hosts
```

OneTA utilizes the simplicity of the Ansible framework and includes the functionality to manage configurations of and deployments to the remote machines in AWS-EC2 environment in Pythonic way. In this thesis project, the ansible-subprocess method was used to trigger the workflow of Ansible.

Furthermore, the Ansible-subprocess is the Python library for Ansible available via PyPI originally developed by MIT to run Ansible dynamically via Python subprocess module [5]. The module was integrated directly as a part of the OneTA library in a way so that it can run the specific Ansible playbook commands. This module was further developed and customized for performing integration and end-to-end testing. Python does everything using this special method. Required parameters needed to be passed in order to construct the playbook command using Python subprocess.

5.9 JSON to YAML conversion

YAML was used to describe the properties needed for AWS-EC2 environment. This is, however, a compulsory component for Ansible for instance creation, machine's configuration and termination of the test VM in AWS environment. To make this solution more dynamic and simple, the creation of Ansible playbook or YAML configuration file for AWS use case was achieved programmatically. Having said that, this was implemented by transforming specific JSON values into YAML format. The following Python code snippet illustrates the approach taken for creating Ansible playbook during AWS configuration management.

```
# Configuration for AWS provisioning
  yaml_file = "aws_tmp.yaml" # New filename to write aws config data
  file = open(yaml_file, 'w')

# Reading AWS specific standard YAML file
  with open('yaml_sample.yaml') as test_file:
    line = test_file.readline()
    while line:
    line = test_file.readline().strip('\t\n\r')
    w_line = line.split(':')
    if len(w_line) == 2 and w_line[1].strip() == "xyz":
      if w_line[0].strip() in data['environment']['aws']:
            w_line[1] = data['environment']['aws'][w_line[0].strip()]
            file.write(w_line[0] + ': ' + w_line[1] + "\n")
        else:
            file.write(line + "\n")
    else:
        file.write(line + "\n")
file.close ()
```

Code Snippet 1: Code snippet for creating Ansible playbook using JSON values

One of the main reasons to implement YAML through JSON is to make the test definition simple and keep the JSON configuration file as a single entry point for the test process. However, from the functional perspective, it was a proper utilization of open-source DevOps tool Ansible. As a result of this, the tester does not need to dive right into complex automation tools or configuration scripts. All it requires is the right parameters that are needed for creating Ansible playbook. In addition, playbooks are more likely to be described as executable documentation. The main OneTA solution script designed in a way so that it can take the inputs from the JSON configuration needed for YAML and capable of creating Ansible playbook during the test run. A temporary YAML file is created in each iteration of the process.

5.10 JSON to manage test configuration in Python

JSON is intended to be a lightweight data-interchange format [21]. OneTA uses a pair of separate JSON files that should be used for entering data needed by the main test script to enable the test orchestration process. A set of example JSON configuration files are included in the main library to define configuration variables in each test scenarios. Writing JSON file is easy, and it is more convenient while dealing with more configuration variables.

The preference of JSON as a configuration input was derived from the organization's standard practice that is predominantly maintained in other test automation solutions. On the other hand, if the tester does not have prior experience with configuration management, it will provide a guideline that the tester can understand. In this solution, the main script loads the configuration values defined in the external file, not in the built-in data structures. Furthermore, configuration in built-in data structure may rise a security issue, especially with secret values, for instance, database information, AWS credentials or other passwords. As for the causes, this issue can be encountered in every place where configuration management is needed. In the worst case scenario, web application or server resources could be compromised if secret values are misused carelessly.

By allowing the configuration data as separate resources can minimize the security issues in general. On the other hand, configurations for different test cases will vary. It is more convenient to describe the test definition in separate configuration files because this way the main script will treat the configuration as just configuration, not as a part of the code. To illustrate, the following section is an example of the JSON configuration that was used to provide the support for AWS:

```
{
  "testcase": {
    "test_name": "OneTA Test",
    "test_set": "Example: av_load_performance.set"
  },

  "environment": {
    "aws": {
      "yaml_file": "./.yaml",
      "ansible_hosts_file": "/etc/ansible/hosts",
      "keypair": "product_test_automation",
      "instance_type": "t2.micro",
      "image":"ami-97e953f8",
      "region": "eu-central-1",
      "aws_access_key": "XXXXXXXXXXXXX",
      "aws_secret_key": "YYYYYYYYYYYYY"
    },
```

Code Snippet 2: Example of JSON for AWS use case

As we can see, JSON is easy for humans to read and write and most importantly a machine can parse easily. In this test automation approach, OneTA only accepts JSON values to feed the test cases. To keep the test definition more specific, it accepts <*config.json*> and <*test.json*> as command-line arguments. The idea behind this is to separate the test environments and test configuration respectively. For example, when the test case needs a simple change tester does not need to modify the whole configuration. Then again, a reference template of JSON files have been included in the main library with the ability to modify using command-line options in order to specify the test cases. A reference configuration provides a template of a proven solution by using a set of preferred execution methods and capabilities.

5.11 Automating workflow with Python

With the help of several other scripts, the main program provides all the required functionalities, such as environment specific logic and execution method to interact with other systems and send the test sets to the desired destination to run the test. OneTA predominantly used Python "subprocess" module for accessing system commands in various conditions, for instance, FSExec in DVMPS and red test automation. This module was used to spawn new processes, connect to their input/output/error pipes, and obtain their return codes.

The libraries and related configuration files used in this solution are structured as follows:

- Test processing assets
    - Execution methods
    - Red TA specific functions
    - Sandbox mock helper
    - Remote execution hooks
- FSExec assets
    - Channel libraries
    - Installer libraries
    - Agent libraries

- Test specification assets
    - JSON files
    - Standard YAML file
- Reference Assets
    - Test files (for future reference)

The above assets are saved into a single directory and usable through the command-line of the Python 3 installed controller machine. To install python packages that are required by the programs into the controller machine, python setup script is included in the root directory of the project. It is used to make the correct installation of the external software and packages. The main test script accepts a maximum of two JSON files as command-line arguments using dynamic loading function and has available options for configuring the JSON inputs. It provides useful help messages using *--help* for a full list of current options.

The command-line arguments are passed as a list of strings, which avoids the need for escaping quotes. In this approach, the Pythonic command-line argument parser "docopt" was included as a solution to create the command-line interface. The docopt module able to generate help and usage messages, therefore, automatically issues errors when users give the program invalid arguments. More information about the docopt implementation method is available here [13]. The following example shows the current implementation of docopt in OneTA solution:

```
Usage: oneta_main.py(-h | --help)
   oneta_main.py - c < file > -i < file > ...
   oneta_main.py - c < file > -i < file > [options]
   oneta_main.py - c < file > -i < file > [--payload = < ipconfig > ]
Options:
   -h--help Show this screen
```

Code Snippet 3: Implementation of command-line interface using docopt

5.12 Jenkins for continuous integration

The Linux based automation server or the test controller machine stated earlier was prepared using the Jenkins slave machine. The Jenkins server is hosted in the green environment and has the special ability to access the Red environment. Having said that, the most important technical aspects it has is the ability to initiate the malware execution in the red cloud environment. This makes it easier to cover all the dependencies and manage test workflows of OneTA. In addition, the controller machine required to fulfil project related dependencies such as Python 3.5, paramiko, docopt, boto3 and Ansible.

In this project, Jenkins paves the way for the standard practice of DevOps in anti-malware product testing and automate the non-human part of the test process. This allows running the tests on the target environments every time new test sets are defined. The actions the controller machine does in the OneTA orchestration process are listed below:

- Allows setting up the system for backend infrastructure
- Copy artifacts from other Jenkins jobs
- Maintain the continuous integration workflows
- Provides real-time monitoring of the test flows
- Allows detecting the errors
- Save the artifacts from successful test execution (artifacts are typically the result of the build process or test outcomes [4])
- Allows to view the test logs

5.13 Running the Tests

The fundamental expectation for this solution was to provide a harmonized and synchronized support with a light-weight and fully featured foundation to interact multiple backend systems from a single configuration. To accomplish this, the primary requirement was to keep the test definition very simple and allow running the test using the command-line interface (CLI). To run the tests, all it requires is to define the directory where the test cases are saved and run the OneTA specific command.

To achieve the simplicity, the JSON-formatted test description defines the flow of the actions during the test execution process. When running with the appropriate arguments, it prints currently configured values and initiates the test process. The executable *oneta_main.py* can be invoked with different combinations of commands, options and positional arguments. Together, these elements form valid syntax for this program and make it usable via command-line. The program accepts specific inputs as arguments to enable the test process in five different environments. A list of available command-line arguments and their description are given in table 6.

Option	Description
--test_name	Name of the test job. For example, the purpose of the test.
--test_set	Name of the test files. For example, which test sets will be used.
--yaml_file	Path of the of the standard YAML file. On the basis of this YAML file, required values will be replaced in the temporary YAML file.
--ansible_hosts_file	Path of the file. The host file is required to construct the Ansible command. The instance will be created on this target host of the AWS environment.
--aws_keypair	Name of AWS key pair. The public key cryptographic key pair is created manually in the target AWS account.
--aws_instance_type	Name of the AWS instance type.
--aws_image	Name of the Amazon machine image.
--aws_region	Name of the Amazon-specific region and availability zone.
--aws_access_key	Security credential for AWS. Highly confidential.
--aws_secret_key	AWS specific confidential item.

--dvmps_base_image	Name of the available image. The image can be chosen from a list of available templates.
--dvmps_expiration	Machine termination time. Should be defined in seconds. After that time the machine will be destroyed.
--comment	Heading of the DVMPS test case.
--dvmps_url	DVMPS specific host name. There is a list of hosts which can be used to create the DVMPS test machine.
--ssh_username	Bare-metal machine's username.
--ssh_host_ip	IP address of the bare-metal machine.
--ssh_pvtkey	Path of the SSH private key of the controller machine.
--ssh_payload	Remote shell or bash commands that should be executed in the bare-metal machine.
--dvmps_payload	A remote shell command that should be executed in the newly created machine in DVMPS environment.
--sandbox_url	<URL> that will be scheduled to analyze
--sandbox_file	<SHA1> or <SHA256> of the file that will be analyzed .
--red_mode	Red cloud-specific item. Mainly used for specifying the test domain, for example, Windows or Linux.
--red_config	Red cloud-specific item. Mainly some Red TA specific Python scripts.
--red_payload	Red cloud-specific scripts, tools, test files etc.

Table 6: List of parameters required for running the test

The test is intended to run from the Jenkins slave machine with the prior setup of the required files in the same place. The test related configuration JSON file can be imported from other resources as well. On the other hand, the default JSON values can be modified from the command-line arguments. The following commands illustrate how the test process can be initiated from the console of the controller machine with the given JSON files having overwriting capability:

```
$ python3 one-ta/src/oneta_main.py -c config.json test.json
Or
$ python3 one-ta/src/oneta_main.py -c config.json test.json --red_payload=tests.py
```

The overall process contains several steps and it maintains specific elements that shown in figure 7.

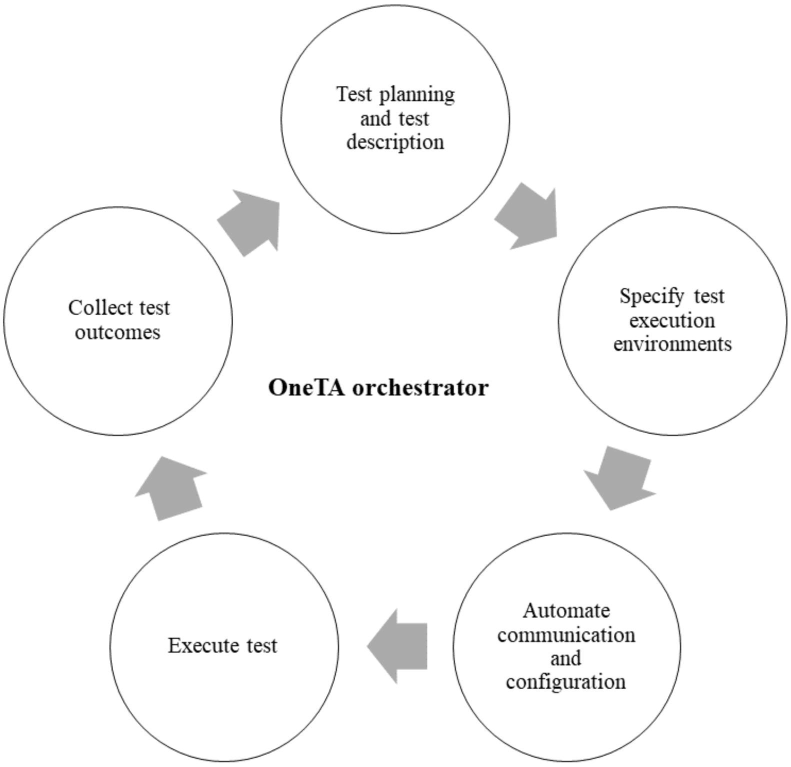

Test planning and test description
Specify test execution environments
Automate communication and configuration
Execute test
Collect test outcomes
OneTA orchestrator

Figure 7: Life cycle of OneTA in anti-malware product test use cases

OneTA has a similar test setup and teardown functionality as other Jenkins related test jobs. The library is managed by VCS tool Git in the bitbucket server. Jenkins use set of bash commands to obtain the library from the VCS and perform several test specific actions to prepare the test. Typically, the test specific settings in Jenkins are prepared by the test managers. That said, OneTA is expected to be used by the test managers as a test execution tool for multiple test cases through continuous integration practice. All it requires is the test specific configuration and test sets and then trigger the "Build" on Jenkins. After the test jobs get finished, Jenkins saves all the related artifacts for future references.

5.14 Packer for machine image creation

One of the optional plans of this thesis project was automatically built machine image for the publicly available cloud environments. The initial plan was to include the crafted machine image for AWS use case. In this case, Packer was considered as an approach to automate the creation of test specific machine image. Having said that, it allows creating identical machine images for different platforms using a single source configuration. The primary intend of this approach was to support the building of crafted images for more platforms, for instance, VMware, VirtualBox, Microsoft Azure, Docker, Google Cloud Engine, etc. and get running machines quickly. Since Ansible was found to be very compatible to install software onto the machine and obtain the needs for AWS use case, as an objective of this thesis, a study was performed to justify the appropriate tool for the OneTA solution.

6 Results: Justification of the Proof of Concept

In the first place, the OneTA solution intends to be applicable to multiple target environments in a single setup. In this thesis scope, some parts of the target environments, for instance, the Red cloud automation with the underlying malware sample storage, were integrated as an existing solution but implemented for the test execution. Additionally, the creation of actual test sets was outside the scope of this thesis project. Each part of the solution was tested and verified against general applicability or known test sets during the development phase. Since the actual solution was intended to develop for internal use of this thesis commissioning organization, the outcomes were justified by the manager of this thesis project.

6.1 Justification of requirements

By implementing the approaches that are thoroughly described in chapter 5, the outcome that was found provided confirmation and evidence that the test execution in multiple environments using simple JSON configuration was successful during the functionality test of this solution. Moreover, the empirical results of this solution were effective against the vendor’s Red Test Automation use cases. The test was performed with known test sets and justified by the manager of this thesis project.

The approach for DVMPS use case was significant in terms of automatic machine provisioning and test execution. It was found that OneTA is more dynamic and useful for DVMPS use case, whereas the earlier approach required some manual interventions. Similarly, the solution for AWS use case utilizing Ansible provides a reliable and effective test execution process. The process is very effective in terms of instant machine creation and deploying related configuration. The core part of the Ansible playbook creation was customized programmatically to support AWS-EC2 machine creation using single configuration and found effective in the test process. Table 7 outlines the currently available supports that have been achieved implementing this solution.

Environments	Current Support
Bare-metal	• Automatic remote connection to the specific machine • Execution of remote commands inside the machine
DVMPS	• Dynamic creation of VM from available templates • Automatic remote connection to the newly created machine • Support for executing remote commands through FSExec • Automatic deletion of the machine
AWS	• Dynamic creation of instance in vendor's AWS environment • Provisioning support for the test instance with Ansible playbook • Remote execution of commands
Sandbox Solution	• Support scheduling the scanning of files and URLs using REST API
Red Cloud	• Support the execution of known test sets in the Red environment

Table 7: Currently available supports for test orchestration through OneTA

In this method, the test requires to follow a specific sequence to avoid the concurrency issues. The sequence was justified against some use cases. Thus, the current sequence was found more reliable and less error-prone. The reliability analysis of the solution was performed against the overall consistency of the workflow and possible outcomes from the test definition. In this test execution process, the test gets triggered to the target environments maintaining that sequence. As for the reference, the sequence was highlighted in figure 6 of chapter 5. Furthermore, the OneTA solution is consistent in executing automated tests as a part of the Jenkins pipeline to produce immediate feedback associated with the target environments.

To justify the OneTA's capabilities, the final outcomes were presented to the members of the development team and managers of the security research team. The actual solution involved proper utilization of known technologies and assembling of existing tools, therefore, harmonizing them programmatically.

6.2 Validity, Reliability and Stability of the concept

The proposed solution intends to support the security research team in the procedure of efficiently incorporating test automation as a practice in the security software testing process of their lab activities. OneTA designed to be minimal in nature, adjustable, consistent and reliable throughout multiple execution processes with an extremely low configuration setup. The solution was developed on top of the available infrastructures, which are utilized by various teams inside the organization for different purposes. The method used in this solution is able to integrate those infrastructures based on prior usage of Python programming language which is a common practice for most of the test related activities in the organization. That also indicates the solution can be used as a common test automation model for present scenarios as well as future scenarios.Besides, the solution can help many stakeholders of the security research unit by automating execution, distribution, and result analysis of the test cases for supporting in-house and public infrastructures. Furthermore, it is useable as a command-line tool which accepts various inputs as arguments with overwriting capability. For the future scenarios, it also has the supporting capability for the pluggable approach with the minimal modification.

6.3 Test cases

The OneTA solution shall cover the functionality of the AV product test against these possible use cases:

a. Run functional test suite for product X on all Windows versions that support
b. Run performance tests on known fixed environment comparing version X with version X-1
c. Check engine X coverage against known malware/known clean sets
d. Run Windows certification tests for product X
e. Run in-house tests for Windows 10 performance requirements
f. Manually run test suite X on product Y on platform Z

6.4 Test coverage and core features

The technique and applicability included in this solution can provide more benefits to the people in this organization who actually write the test cases. This includes Developers, Test Engineers, Operation Engineers, Malware Analysts, Security Researchers and so on. The advantage includes an improvised way of conducting different test processes in the same pipeline; thus, reducing the complexity of maintaining different tools and resources. As a result of this, many test environments came under automated support from operational aspects with this basic refactoring. To summarize, the current OneTA solution includes the following features:

- Follows pure DevOps strategies
- Applicable in different test scenarios
- The library is usable via command-line
- Test definition is customizable via command-line options
- Supports parsing multiple config files in single command
- Full test coverage for five different environments
- Usable in the Jenkins environment
- Capable of provisioning a new test machine in AWS and DVMPS
- Supports sandbox API to schedule analysis
- Supports remote logging, file copying and executing commands through FSExec in newly provisioned machine
- Zero external dependencies of the core libraries
- Test coverage is expandable for future pluggable environments

- Test outputs are viewable through Jenkins console
- Test Artifacts can be retrieved from Jenkins server
- Integration of Ansible can support more cloud platforms
- Supports execution of repeated test cases
- Aids in testing a large test matrix
- Supports execution of repeated test cases
- Saves test preparation time

7 Conclusion

Initially, the idea of this orchestration solution was proposed as a concept, but never implemented in this thesis commissioning organization. The plan was to design and implement a solution that satisfies the fundamental requirements for different cloud-based systems and automation tools, then integrate it into a single framework. Several solutions for test execution were already available, but the security research team wanted to have their own variants. This study was the first step to go some way towards enhancing some known parts of the existing solution and expand the current test coverage in the continuous integration practice. After identifying problems with widespread range of in-house test environments and tools, the thesis addressed a solution focusing on network level automation in the process of anti-malware product test.

However, test automation has been proposed as a solution, but the available tools and techniques experience a lack of general applicability. The scope of the thesis mainly consisted of automating the test process and analysis. The project demands a research on internal infrastructures, different execution methods and existing test automation processes. During the implementation of the plan, a broad analysis of the target environments including related tools and technologies and existing test cases was performed intensively to maintain the industry standard practices. The proper solution involved identifying the right automation tools for infrastructure provisioning, implementation method for the in-house test execution process, developing scripts for preparing test environments and simplification of test the definition.

Adopting a new test execution infrastructure and automating the process is not easy due to lack of information, knowledge and skills and typically it requires a plan that spans people, process, and technologies [44]. On the other hand, the main difficulty in the management of infrastructure involves communication between different stockholders inside the organization. For developers and testers, it is a common problem to suffer from project complexity and repetitive manual process. However, we can prepare services by hand, for example, setting up the SSH connections to each one, modifying config files, installing required packages and so on. Performing these tasks are not only tedious but also time-consuming, therefore, it leads to encounter errors. Furthermore, admins of each system need to find one advent of good CI solution and configure them accordingly. In addition, there are a variety of testing tools, ranging from free and open-source tools that support different testing types and technologies. Also, organizations write software to support customizing or integrating other software or solution into internal IT systems. These create more dependencies among many teams inside the organization.

Each tool tends to support particular situations. The selection of an appropriate testing tool to satisfy the needs could be one of the big challenges in the test automation process. Plus, in many cases, developers do not conduct enough research before deciding on tool selection. Some workarounds are often made to tackle particular use cases. These scenarios emphasize the need for a modular solution in a single namespace. In this case, the current solution tried to enable people inside the organization without let them emphasizing how the network communication establishes in different environments to execute the tests and thereby offers comprehensive guideline that can easily be applied to perform various types of tests in a single process.

The challenge is that significant effort is needed in designing a test process that will capitalize on the potential for improvement that is offered by many automation tools. Producing this kind of solution not only requires experienced engineers, but also IT resources, which are subject to constraints such as time, communication, and expertise. OneTA focused to provide a harmonized solution for all related components so that the cross-system requirements are fulfilled for different environments. The work has proved that these requirements can be fulfilled by applying systematic DevOps approach. Nevertheless, the thesis successfully developed a minimum viable solution based on the requirements, which were set by the managers of the security research team to overcome an in-house test automation challenge. Thus, it encouraged applying a more programmatic approach to bring the test automation solution into reality. Moreover, there was proper utilization of Python programming language for the test automation purpose.

The actual work targeted testing of a new possibility and envision for software engineering teams by developing something new that solves several test related problems.

A fully functional test automation solution for target infrastructures was the base for this proof of concept (PoC). The project or the solution itself concentrated on DevOps or more specifically DevSecOps approach so that it can collaborate with the security product development and operation teams. The final outcome provides a significant usefulness and indicates that by utilizing OneTA solution, the security research team can boost efficiency, cut dependency and help other teams flourish better. However, justification of usable technologies and tools for target environments as functional and operational requirements were mainly made with the consultation of senior engineers of this thesis commissioner organization.

On logical grounds, there is no compelling reason to argue that antivirus tests need better methodology. There might be controversies about whether we should promote test automation in anti-malware product testing activities or not. From where I stand, test automation might have a huge payback, and it should not be forgotten that test automation is nowadays dominating in agile development context and it has received much attention in the last few years. Many test automation projects have a proven record of successes when people are creative and able to overcome the challenges effectively [35]. Needless to say, the next decade is likely to see a considerable rise of DevOps in the software development process where cloud-native approach will play a vital role.

7.1 Limitations and suggestions for future work

Many different test cases and experiments have been left for the future work due to a lack of time. Until now, the outcomes are promising and validated by a couple of use cases. Since the validity of this solution was performed mainly against a minimum number of use cases, further work needs to be done to establish the justification of whether the solution is consistent in actual scenarios. Future work should concentrate on justifying the solution against actual payload.

The current solution only allows performing the test against the configuration for five different environments. The solution is valid for a specific sequence of task execution. Error handling rules were not implemented in this case yet. This is an important issue for future considerations. OneTA solution should provide support for conditional execution of tasks. The selection of the test environments should be considered as future implementation to make the solution more useful.

As of now, the test consistency has been checked against currently supported environments, and it was performed against simple use cases. In the actual scenario, when lengthier test cases will be performed, performance may not be the same. The performance evaluation of the OneTA solution should be considered as a future study.

As we know, Ansible supports many cloud platforms as a configuration management tool. Thus, it creates more scope to integrate other cloud platforms such as vCloud, Microsoft Azure, Google Cloud, etc. as pluggable environments. The solution can be extended to support more cloud platforms with a similar approach. The current implementation will serve as a base for future integration of other cloud platforms. In future, OneTA should target adding more cloud environments. Besides, further development could be undertaken in the following areas:

- The execution of all test cases at the same time
- Proper JSON schema should be prepared for a test definition
- Integration of Packer for automatic image creation through test definition
- Test related logic can be improved in the main Python script

References

[1] R. Anderson, “From Bare Metal to Private Cloud: Introducing DevSecOps and Cloud Technologies to Naval Systems,” M.S. Thesis, Dept. of Softw. Eng., Auburn Univ., Alabama, USA, 2018.

[2] S. Amaricai and R. Constantinescu, “Designing a Software Test Automation Framework,” Informatica Economica, vol. 18, no. 12014, pp. 152-161, Jan. 2014.

[3] “API Mocking,” Soapui.org, 2018. [Online]. Available: https://www.soapui.org/learn/mocking/what-is-api-mocking.html [Accessed: 03- Mar- 2018].

[4] “Artifact-JenkinsAPI 0.2.26 documentation,” Pythonhosted.org, 2018. [Online]. Available: https://pythonhosted.org/jenkinsapi/artifact.html [Accessed: 11- Apr- 2018].

[5] “ansible-subprocess,” PyPI, 2018. [Online]. Available: https://pypi.org/project/ansible-subprocess/ [Accessed: 11- Apr- 2018].

[6] S. Berner, R. Weber and R. K. Keller, “Observations and lessons learned from automated testing,” in Proc. of the 27th Int. Conf. on Softw. Eng., St. Louis, NY, USA, pp. 571-579, May 2005.

[7] J. Brodkin, “With long history of virtualization behind it, IBM looks to the future,” Network World, 2009. [Online].
Available: https://www.networkworld.com/article/2254433/virtualization/with-long-history-of-virtualization-behind-it--ibm-looks-to-the-future.html [Accessed: 16- Mar- 2018].

[8] B. Dijkstra, “A guide to automation testing frameworks and how to build yours,” TechBeacon, 2018. [Online]. Available: https://learn.techbeacon.com/units/guide-automation-testing-frameworks-how-build-yours [Accessed: 16- Apr- 2018].

[9] A. Chatterjee, “How to Achieve DevOps Through Automation,” dzone.com, 2016. [Online]. Available: https://dzone.com/articles/what-is-devops-and-how-automation-helps-achieve-it [Accessed: 18- May- 2018].

[10] D. Firesmith, “Common Testing Problems: Pitfalls to Prevent and Mitigate,” insights.sei.cmu.edu, 2013. [Online].
Available: https://insights.sei.cmu.edu/sei_blog/2013/05/common-testing-problems-pitfalls-to-prevent-and-mitigate-1.html [Accessed: 23- Mar- 2018].

[11] C. Conde and N. Attila, “Development and Test on Amazon Web Services,” amazonwebservices.com, 2012. [Online].
Available: https://media.amazonwebservices.com/AWS_Development_Test_Environments.pdf [Accessed: 13- May- 2018].

[12] “Cuckoo Sandbox- Automated Malware Analysis,” cuckoosandbox.org, 2018. [Online]. Available: https://cuckoosandbox.org/ [Accessed: 27-May- 2018].

[13] “docopt-language for description of command-line interfaces,” docopt.org, 2018. [Online]. Available: http://docopt.org/ [Accessed: 09-Apr- 2018].

[14] A. Singh, “DevSecOps: Integrating Security into DevOps,” Algoworks, 2018. [Online]. Available: http://www.algoworks.com/blog/devsecops-integrating-security-into-devops/ [Accessed: 11-Jun- 2018].

[15] C. Ebert, G. Gallardo, J. Hernantes and N. Serrano, "DevOps," IEEE Software, vol. 33, no. 3, pp. 94-100, Jun 2016.

[16] M. Fewster and D. Graham, “Software Test Automation,” New York, NY, USA: Addison-Wesley Professional, pp. 04-25, 1999.

[17] D. Linthicum, “DevOps tools best practices: A 7-step guide,” TechBeacon, 2016. [Online]. Available: https://techbeacon.com/7-steps-choosing-right-devops-tools [Accessed: 01-Mar-2018].

[18] L. Hochstein and R. Moser, “Ansible: Up and Running,” California, USA: O'Reilly Media, pp. 01-08, 2017.

[19] “Instances and AMIs - Amazon Elastic Compute Cloud,” docs.aws.amazon.com, 2018. [Online]. Available: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-instances-and-amis.html [Accessed: 06-Jun- 2018].

[20] R. Jabbari, N. Ali, K. Petersen and B. Tanveer, “What is devops? A systematic mapping study on definitions and practices,” in Proc. of the Scientific Workshop Proc. of XP2016, Edinburgh, Scotland, pp. 12-13, May 2016.

[21] “JSON,” json.org, 2018. [Online]. Available: https://www.json.org/ [Accessed: 07-Apr-2018].

[22] Y. Koh, R. Knauerhase, P. Brett, M. Bowman, Z. Wen and C. Pu, “An analysis of performance interference effects in virtual environments,” in Proc. of the 2007 IEEE Int. Symposium on Perform. Analysis of Syst. & Softw. San Jose, CA, USA, pp. 200-209, May 2007.

[23] E. Kim, J. Na and S. Ryoo, “Test automation framework for implementing continuous integration,” in in Proc. of 2009 Sixth Int. conf. on Inf. Technol.: New Generations, Las Vegas, NV, USA, pp. 784-789, Jun. 2009.

[24] L. Malhotra, D. Agarwal and A. Jaiswal, "Virtualization in Cloud Computing," Journal of Inf. Technol. & Softw Eng., vol. 04, no. 02, 2014.

[25] “Mocking External APIs in Python,” realpython.org, 2018. [Online]. Available: https://realpython.com/testing-third-party-apis-with-mocks/ [Accessed: 25-May-2018].

[26] K. McMeekin, “Test Automation vs. Automated Testing: The Difference Matters,” QASymphony, 2017. [Online]. Available: https://www.qasymphony.com/blog/test-automation-automated-testing/ [Accessed: 27- May-2018].

[27] G. Narcisi, “8 AWS Offerings Gaining Popularity Right Now,” crn.com, 2018. [Online]. Available: https://www.crn.com/slide-shows/cloud/300099476/8-aws-offerings-gaining-popularity-right-now.htm [Accessed: 21-Jun-2018].

[28] M. Nabil, “The Automation Testing and Agile,” medium.com, 2017. [Online]. Available: https://medium.com/@Moatazeldebsy/the-automation-testing-and-agile-7a8a8c983ed0 [Accessed: 26-Feb-2018].

[29] “Open source software for creating private and public clouds,” openstack.org, 2018. [Online]. Available: https://www.openstack.org/ [Accessed: 28-Apr-2018].

[30] R. Peltonen, “Automated Testing of Detection and Remediation of Malicious Software,” M.S Thesis, Dept. of Info. Tech. Eng., Helsinki Metropolia Univ. of Applied Sci., Helsinki, Finland, 2017.

[31] M. Russinovich, “PsExec- Windows Sysinternals,” docs.microsoft.com, 2016. [Online]. Available: https://docs.microsoft.com/en-us/sysinternals/downloads/psexec [Accessed: 18- Apr-2018].

[32] “Parallel Worlds: Agile and Waterfall Differences and Similarities,” Software Engineering Institute- Carnegie Mellon University, Massachusetts, USA, Oct. 2013. [Online]. Available: http://www.dtic.mil/dtic/tr/fulltext/u2/a610501.pdf [Accessed: 24-Mar-2018].

[33] “2016 State of DevOps Report Puppet,” puppet.com, 2017. [Online]. Available: https://puppet.com/resources/whitepaper/2016-state-devops-report/thank-you [Accessed: 07-Jun-2018].

[34] “The Python Package Index,” pypi.org, 2018. [Online]. Available: https://pypi.org/ [Accessed: 19-May-2018].

[35] B. Pettichord, “Success with Test Automation,” in Proc. of the Ninth International Quality Week, San Francisco, CA, USA, pp. 02-07, May 1996.

[36] M. Rajkumar, A. Pole, V. Adige and P. Mahanta. “DevOps culture and its impact on cloud delivery and software development,” in Proc. of the Int. Conf. on Advances in Computing, Communication, & Automation. (ICACCA’16), Dehradun, India, pp. 01-06, Sep. 2016.

[37] V. Roy, “Top 10 Best DevOps video tutorials | Learn DevOps step by step,” topzenith.com, 2018. [Online]. Available: https://www.topzenith.com/2018/02/top-10-best-devops-video-tutorials.html [Accessed: 25-Feb-2018].

[38] L. Riungu-Kalliosaari, S. Mäkinen, L. E. Lwakatare, J. Tiihonen and T. Männistö, “DevOps adoption benefits and challenges in practice: a case study,” in Proc. of Int. Conf. on Product-Focused Softw. Process Improvement, Trondheim, Norway, pp. 590-597, Nov. 2016.

[39] M. Rouse, “What is sandbox?,” WhatIs.com, 2018. [Online]. Available: https://searchsecurity.techtarget.com/definition/sandbox [Accessed: 14-May-2018].

[40] “Skillsets to Work In DevOps Environment- A Comprehensive Guide,” DevopsQube, 2018. DevopsQube. [Online]. Available: https://devopscube.com/skillsets-to-work-in-devops-environment/ [Accessed: 19-Mar-2018].

[41] K. Sadhukhan, R. A. Mallari and T. Yadav. “Cyber Attack Thread: A control-flow based approach to deconstruct and mitigate cyber threats,” in Proc. of 2015 Int. Conf. in Computing and Network Communications (CoCoNet), Trivandrum, India, pp. 170-178, Feb. 2015.

[42] J. Smeds, K. Nybom and I. Porres, “DevOps: a definition and perceived adoption impediments,” in Proc. of Int. Conf. on Agile Softw. Develop, Helsinki, Finland, pp. 166-177, May 2015.

[43] “Sandboxing- Cuckoo Sandbox v2.0.6 Book,” cuckoo.sh, 2018. Available: https://cuckoo.sh/docs/introduction/sandboxing.html [Accessed: 19-Mar-2018].

[44] S. Sharma, “What is DevOps?,” in DevOps for Dummies, 3rd ed., New Jersey, USA, John Wiley & Sons, pp. 03-17, 2017.

[45] J. Turnbull, “The Docker Book,” dockerbook.com, 2014. [Online]. Available: http://opisboy.bandungbaratkab.go.id/books/James.Turnbull.The.Docker.Book.Containerization.is.the.new.virtualization.B00LRROTI4.pdf [Accessed: 29-Jun-2018]

[46] M. McLoughhlin, “The QCOW2 Image Format,” people.gnome.org, 2008. [Online]. Available: https://people.gnome.org/~markmc/qcow-image-format.html [Accessed: 28- Mar-2018.

[47] “What Can Enterprises Expect from DevOps In 2018? – Powered by Algoworks,” Medium.com. 2018. [Online]. Available: https://medium.com/all-technology-feeds/what-can-enterprises-expect-from-devops-in-2018-5694461bf44f [Accessed: 23-Mar-2018]

[48] A. Hajda, “Winexe,” kali.org, 2014. [Online]. Available: https://tools.kali.org/maintaining-access/winexe [Accessed: 18-Apr-2018].

[49] “What does IT Infrastructure mean?,” Techopedia.com, 2018. [Online]. Available: https://www.techopedia.com/definition/29199/it-infrastructure [Accessed: 21-Jun-2018].

[50] J. Wettinger, U. Breitenbücher, O. Kopp and F. Leymann, “Streamlining DevOps automation for Cloud applications using TOSCA as standardized metamodel,” in Future Generation Computer Systems, pp. 317-332, Mar 2016.

[51] “What is KVM?,” RedHat, 2018. [Online]. Available:
https://www.redhat.com/en/topics/virtualization/what-is-KVM [Accessed: 28-Jul-2018].

[52] “Working with Playbooks,” Ansible Documentation, 2018. [Online]. Available: https://docs.ansible.com/ansible/2.5/user_guide/playbooks.html [Accessed: 03-Jul-2018]

[53] J. Giménez, “What Is Jenkins and Why Should You Be Using It?,” bugfender.com, 2017. [Online]. Available: https://bugfender.com/blog/what-is-jenkins-and-why-should-you-be-using-it/ [Accessed: 25-Jun-2018].

[54] “Paramiko,” Paramiko.org, 2018. [Online]. Available: http://www.paramiko.org/ [Accessed: 21-May-2018].

[55] “What is Devops? What does it really mean?,” DevOpsQube, 2016. [Online]. Available: https://devopscube.com/what-is-devops-what-does-it-really-mean/ [Accessed: 22-Feb-2018].

[56] T. Ylonen and C. Lonvick, “The secure shell (SSH) protocol architecture,” No. RFC 4251, 2006. [Online]. Available: https://www.ietf.org/rfc/rfc4251.txt. [Accessed: 23-Jul-2018].

[57] S. Zohrah, “How to choose the right DevOps tools,” atlassian.com, 2016. [Online]. Available: https://www.atlassian.com/blog/devops/how-to-choose-devops-tools [Accessed: 15-Apr-2018].

[58] M. Sumrell, "From Waterfall to Agile - How does a QA Team Transition?," In Agile Conf. Agile 2007, IEEE, Washington, DC, USA, pp. 291-295, Aug. 2007.