Utilize Docker container and Selenium Grid technology to improve resource efficiency in the runtime of a Software Deployment & Testing Framework (SDTF).

[Document subtitle]

VASCO Data Security

[Course title]

# Preface

# Summary

# Table of contents

[Preface 1](#_Toc513560894)

[Summary 1](#_Toc513560895)

[Table of contents 2](#_Toc513560896)

[List of figures, graphs 4](#_Toc513560897)

[Introduction 5](#_Toc513560898)

[About the webadmin suite 5](#_Toc513560899)

[Problem statement 7](#_Toc513560900)

[Current situation 7](#_Toc513560901)

[Objectives 8](#_Toc513560902)

[# Methodology (practical goals) 8](#_Toc513560903)

[Technologies 9](#_Toc513560904)

[Selenium 9](#_Toc513560905)

[Selenium grid 9](#_Toc513560906)

[Docker 9](#_Toc513560907)

[What is a container? 9](#_Toc513560908)

[Why containers? 11](#_Toc513560909)

[Implementation 13](#_Toc513560910)

[Proposed architecture 13](#_Toc513560911)

[Changes needed 13](#_Toc513560912)

[Solution 14](#_Toc513560913)

[Deployment architecture 14](#_Toc513560914)

[The containers 14](#_Toc513560915)

[Impact on the existing WebAdmin 15](#_Toc513560916)

[Changes needed to the WebAdmin Suite 15](#_Toc513560917)

[Challenges 16](#_Toc513560918)

[IE in a container? 16](#_Toc513560919)

[Issues with the IE node 16](#_Toc513560920)

[Performance difference 17](#_Toc513560921)

[Conclusion 18](#_Toc513560922)

[Appendix 19](#_Toc513560923)

[IAS on vCloud 19](#_Toc513560924)

[Dockerfiles 21](#_Toc513560925)

[For use with the SSH tunnel 21](#_Toc513560926)

# List of figures, graphs

# Introduction

VASCO Data Security is a company mainly active in cyber security, mostly known for two factor authentication services and electronic signature software.

(list products?)

This thesis is written in context of Quality Assurance (QA). The QA team utilizes a Software Development and Testing Framework (SDTF) to practice system testing on the different products that are being developed.

One of the products, the IDENTIKEY Authentication Server (IAS), is a centralized platform designed to deliver authentication lifecycle management via a single integrated system, supporting a multitude of other technologies and VASCO products. This server is managed via a web based platform, “IAS Web Administration”.

This product is in continuous development and thus, needs to be tested frequently, which is why the QA team has a set of tests in place that automates testing the web application, supported by the SDTF.

The SDTF is a system testing framework written in Python and based on the python unit test library “PyUnit”. The purpose of the framework is to automate the system testing of an application or service, and see if the output matches the expected output for certain inputs. The SDTF automates this process and sends back feedback to a test management system, in this case TestLink.

This thesis is going to focus on improving the runtime of the SDTF for the WebAdmin test suites.

Other test suites in the SDTF include

* IDENTIKEY administration server
  + Build intake
  + Installation
  + Generic authentication
  + Radius authentication
  + Soap authentication
  + Tcl administration
  + Validation
  + *Webadmin*
* Digipass
* Dpscreader
* TID
* Native bridge

The SDTF runs on a test control host (TCH) running Windows

## About the webadmin suite

The webadmin test suite is a collection of 235 test cases divided between 20 test suites:

1. Logout
2. Logon and Privileges
3. Create User
4. Import Users
5. List Users
6. Search User, Find Menu
7. Search User Find/Manage Menu
8. Manage Admin Privileges
9. Manage Dashboard
10. Manage User Account
11. Manage User Attributes
12. Manage User Info
13. Manage Virtual DIGIPASS
14. Manage Assigned DIGIPASS
15. Import DIGIPASS
16. Search DIGIPASS, Find Menu
17. Search DIGIPASS, Find/Manage Menu
18. Manage DIGIPASS
19. Assign DIGIPASS
20. Manage DIGIPASS Application

Each test suite exists of a base case, of which the other cases in the suites are variations, aside from a few exceptions in each suite.

# Problem statement

## Current situation

\\srv-be-file.vasco.com\UserProfiles$\willaar1\Documents\Stuff voor thesis\thesisDocs\Presentaties\prentjes\Poging2000.emf

Today the SDTF is run on virtual machines on vCloud running windows, with all the dependencies installed, as well as all the third party software needed for some tests.

The IAS is also deployed in the same way; on a VM on vCloud, with all the supporting services necessary, such as a database and the IAS Web Administration service.

However, this deployment strategy is very resource intensive, as with each test you want to run you need to boot up a Test control host, as well as a machine running IAS. This is made evident by the fact that vCloud often gets saturated by the amount of machines that are running on it.

<Include execution time and resources>

# Objectives

The goal of this research project is twofold:

The primary goal is to come up with a method to reduce the runtime of the tests in the Software Development and Testing Framework to decrease the latency between starting the test suite and getting the results. The focus of how this is achieved is going to be on the WebAdmin test suite. As the conclusions made for this test suite are assumed to also be applicable to other test suites.

The WebAdmin suite is designed run against different browsers such as Google Chrome, Firefox and Internet Explorer. In the current situation these test are run sequentially for each browser. For example, first Firefox then Chrome. The idea that is going to be explored is whether these suites can be run in parallel, as this would greatly decrease the runtime.

Another question that comes up when talking about parallelization is if the suites can be run against the same environment at the same time? And what changes need to be made to the test suites to be able to achieve this goals. This ties in with our second point.

The second objective is to research how to reduce the resource usage of these tests. This increases the overall efficiency of the environment, and allows for more tests to be run concurrently on the same platform.

A path that is going to be explored is a new deploy strategy for the test control hosts, as they now each require a virtual machine to be started up, with all the requirements to run all the tests in the SDTF.

## # Methodology (practical goals)

# Document the advantages and disadvantages of the proposed deployment architecture

# Include a discussion on the difficulties encountered with the solution/technologies

# Quantify the proposed solution’s impact in terms of saved resources and execution time

# Technologies

Discussing the technologies utilized to work towards the end goal: a more efficient system

## Selenium

Selenium is a portable software testing framework for web applications. Selenium provides a record/playback tool for authoring tests without learning a test scripting language (Selenium IDE). It also provides a test domain-specific language (Selenese) to write tests in a number of popular programming languages, including C#, Java, Groovy, Perl, PHP, Python and Ruby.

The tests can then be run against most modern web browsers. Selenium deploys on Windows, Linux, and Macintosh platforms. It can be used for unit, regression, smoke, integration and acceptance testing.

In this specific test suite, we make use of Selenium Webdriver. This service accepts commands, sends them to the relevant browser through a browser-specific browser driver, and then retrieves the results. Most browser drivers launch and access the browser application directly.

### Selenium grid

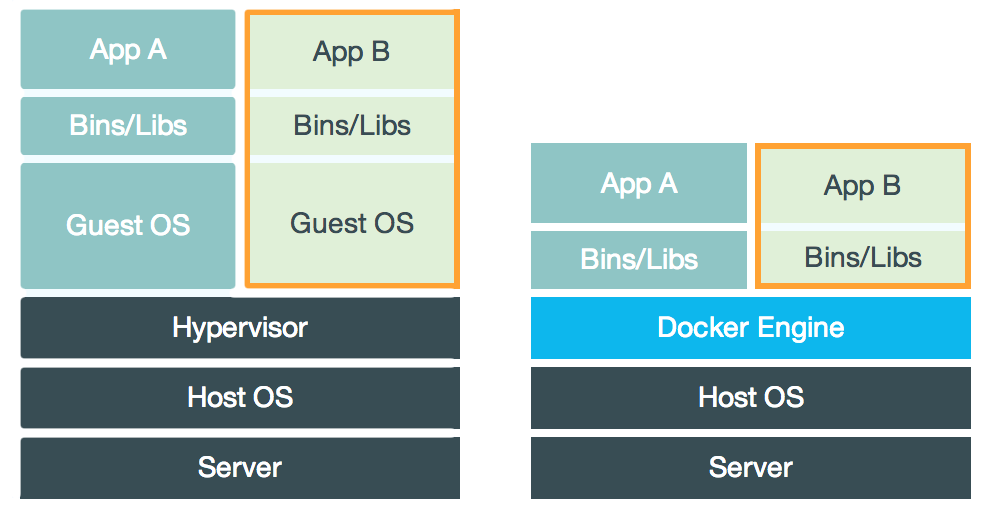
Selenium Grid is a framework that allows you to run tests on different machines against different browsers in parallel. I.e.: running multiple tests concurrently against potentially different machines running different browsers and operating systems. In short it allows you to run tests in a distributed test execution environment.

The grid consists of one or more worker machines called nodes and a single “master” machine called a hub. Someone who uses this service for testing would send a test to the hub, alongside with information on which browser, and if applicable, which operating systems to run the test on. The hub knows the configuration of each node that is register at the hub, and selects the node with the correct attributes. Once a node has been selected, Selenium commands initiated by the test are sent to the hub, which passes them through to the node assigned to that test. On this node, a browser is opened and executes the selenium commands within that browser against the application under test.

## Docker

### What is a container?

Maybe make an original diagram comparing a vm and a container



The best way to explain what a container is, is probably comparing it to a virtual machine (VM for short). While containers and VMs are two different technologies, they can be used for a similar purpose: the isolation of an application and the relevant dependencies in a self-contained unit.

Furthermore, utilizing VMs or containers this way can remove the need for physical hardware, and can result in a more efficient use of resources

#### Virtual Machines

At its core, a virtual machine is a complete emulation of all the hardware components that make up a computer that runs an operating system, executes commands and runs applications, as if it “were a physical” vs. “would be an actual” computer. A VM, or multiple VMs, can runs on a single, or distributed set of machine(s) through a hypervisor. A hypervisor can run either on the operating system of host machine (a hosted hypervisor) or directly on the hardware (native or bare-metal hypervisor)

The hypervisor itself runs on a physical system called a “host machine”. The host machine provides the VMs with the necessary resources, like RAM and CPU. These resources can be divided amongst the VMs to best suite your use-case.

A defining feature of a VM is that they come with a complete virtualized hardware environment, which includes a CPU, RAM, storage, network adapters, graphical interfaces, and more.

#### Containers

Unlike a VM which provides hardware virtualization, a container provides virtualization on the operating system level. It does this by abstracting the user space.

A container is similar to a VM in quite a couple of ways. For instance, they have a private process tree, their own network interface and IP address, can execute commands as root and mount filesystems.

The big difference with virtual machines is that containers package up just the user space, and not the kernel or virtual hardware like a VM does. A container shares the host system’s kernel with other containers. But, because every container gets its own user space, it is possible to run multiple containers on a single host.

### Why containers?

While virtual machines are more robust than containers, there are a couple of reasons why it makes more sense to use containers.

#### Image size

In contrast to VMs, Docker images tend to be very small, a couple hundred megabytes, maybe even tens. While images for VMs tends to be upwards of twenty gigabytes. This is because contrary to a container, there is way more overhead involved, such as the operating system, hardware, dependencies, and so on; while a container only contains a minimal operating system and the application you want to run on it, as well as any dependencies it might have.

However, this difference is going to be less drastic in this implementation, because the application we are running is quite large, and needs a good few dependencies. Nevertheless, the size of the image that runs SDTF should not exceed thee gigabytes, which is still a significant decrease.

#### Flexibility

Another great boon for containers is their start-up speed. Once you have built a container image, you can start up a container fairly quickly, in a matter of seconds even. Starting up a VM Takes way longer than that, and could even take a couple of minutes. This time gain is probably negligible in the bigger picture of this project, as it is only necessary to start up the container once before you run the tests, and then it runs for a couple hours.

Utilizing Docker Compose, it is possible to bundle multiple container images together in a service. Doing this provides a couple of interesting features, but more on that later.

The most relevant use case of Docker Compose is that it allows you to bundle the containers into one service for easy set up and tear down of the testing infrastructure. By defining the full environment in a Compose file, you can create and destroy these environments in just a few commands:

$ docker-compose up (this starts up the service)

<run the tests>

$ docker-compose down (this destroys the service)

This is way more convenient then setting up the different containers to run individually.

#### Resource usage

Because a Docker container does not need to simulate any hardware, and has a more minimal version of an operating system, it uses way less processing power and RAM to function. This results in a meaningful decrease in resource overhead that is going to be decisive for increasing the efficiency of the SDTF.

Maybe something about unionfs here?

#### Summary

Considering all the above it makes sense to move from the current model to a model that uses Docker container to instantiate the services required for unit-testing the Webadmin portion of the Identikey Authentication Server.

# Implementation

## Proposed architecture

Needed: a picture that explains Selenium (grid)

\\srv-be-file.vasco.com\UserProfiles$\willaar1\Documents\Stuff voor thesis\thesisDocs\Presentaties\prentjes\Poging1000.emf

## Changes needed

Changes needed to the test suite code to make it run on the new architecture

# Solution

Walkthrough of the steps taken to accomplish the research goal

## Deployment architecture

As discussed in a previous chapter, there is a theoretical advantage to using Docker containers when it comes to performance and resource usage when compared to virtual machines. Mostly due to their lightweight nature, both in computing resources as image size.

In this chapter, it is explored if we can leverage this advantage to benefit the resource usage of the environment

### The containers

#### Container with SDTF as TCH

FROM python:2.7

ADD python\_test.py /

# Selectively add parts of SDTF needed for IAS testing

ADD sdtf /root/sdtf

RUN apt-get update && apt-get install -y \

python-tk \

python-pip \

python-ldap \

&& rm -rf /var/lib/apt/lists/\*

RUN apt-get update --fix-missing

RUN pip install virtualenv

RUN chmod +x /root/sdtf/install\_sdtf\_linux.sh

RUN /root/sdtf/install\_sdtf\_linux.sh

ENV LMPW=U89yMN8R8bwgOKR1k6tz

CMD [ "/bin/bash" ]

.dockerignore:

sdtf/doc

sdtf\_less\_old

sdtf\_old

sdtf/\*untime

sdtf/\*-activate

sdtf/\*-activate.bat

sdtf/\*-deactivate.bat

sdtf/specs

sdtf/suites/digipass

sdtf/support

sdtf/test

sdtf/webadmin

sdtf\_copyfromwindows/

Basicly just add all directories that you want the Docker daemon to skip when building the image. By default, the whole working directory gets passed to the Docker daemon when you use

docker build -t sdtf/sdtf\_3 .

As specified by the “.”. Note that the path you specify also needs to contain the Dockerfile and the .dockerignore.

#### Selenium Grid

#### Docker-Compose

## Impact on the existing WebAdmin

## Changes needed to the WebAdmin Suite

# Challenges

Discussing technological limitations and other problems

## IE in a container?

For the execution of web based test suites, there is a need for the tests to be run on Internet Explorer to ensure a good coverage of use cases. There are, however, a few issues with the proposed deployment architecture related to Internet Explorer. The first being: running internet explorer in a container

The SeleniumHQ repository offers Docker images for a Selenium Hub and for nodes with either Firefox or Google Chrome, with Internet Explorer being notable absent.

There is, however, a NodeBase dockerfile provided to serve as a basis from where to build your own dockerfile to create a non-standard image. The NodeBase is equipped with only the install of selenium, and does not include a browser.

However, the issue here is that the NodeBase image is based on the Base image in the repository that in turn is based on Ubuntu 16.04. In addition, Internet explorer was never ported to Ubuntu, or any other Linux distribution for that matter, so installing it on the NodeBase image is practically impossible.

There is always the possibility of installing wine (a compatibility layer capable of running windows applications on other operating systems such as Linux and macOS), but that would be a bad idea as running in on a compatibility layer in a container seems like more trouble than it’s worth. Especially considering that Internet Explorer is already known for running slowly and being generally unstable on wine.

Another problem is that if you were to build a Docker image based on Windows and run an Internet Explorer browser on it that way, then you wouldn’t be able to run that image on a Linux host. This is because a container image always needs to be based on the same OS as the host you are running it on.

The workaround here is just to run a selenium node running Internet Explorer on a windows machine and adding it to the grid. This way we lose the scalability of running it as a container, but it is necessary to get it working.

(do I need to put the steps to configure a machine to be a selenium node here?)

## Issues with the IE node

Discuss the IE Webriver and why it makes it difficult/impossible to run the suite

<https://stackoverflow.com/questions/19662045/selenium-hover-elements-with-ie>

<http://jimevansmusic.blogspot.be/2013/01/revisiting-native-events-in-ie-driver.html>

<https://github.com/SeleniumHQ/selenium/wiki/InternetExplorerDriver>

<https://github.com/seleniumhq/selenium-google-code-issue-archive/issues>

# Performance difference

# Conclusion

Should at least answer the following questions:

* Which parts of the proposed deployment architecture were actually implemented?
* Recap which other parts were scrapped (maybe shortly touch on the why), and which parts were changed due to technological limitations or other constraints?
* What didn’t work at all?
* Is there a gain in efficiency?
  + Why, or why not?
* Is the gain worth the trouble?

Include suggestions for future research

NO NEW INFORMATION HERE!!!

# Appendix

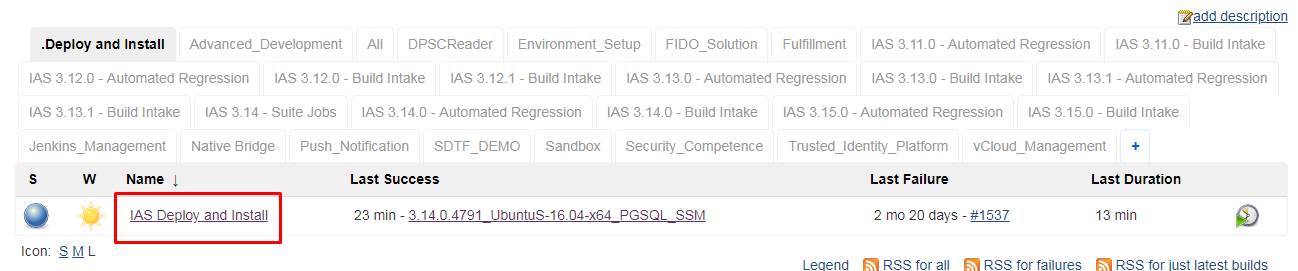
## IAS on vCloud

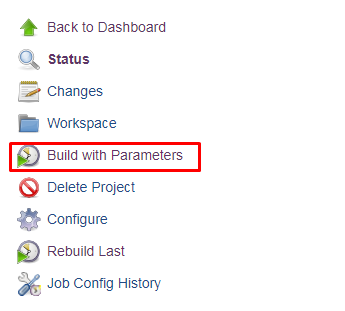
This step describes how to set up an Identikey Administration

<http://jenkins-qa.vasco.com/> and log in with AD username and pw

click on deploy and install







Test project is customizable (I just left it default)

Test plan needs to be SDTF\_DEMO (leave default)

AD\_USER is your username

AD\_PW is your password

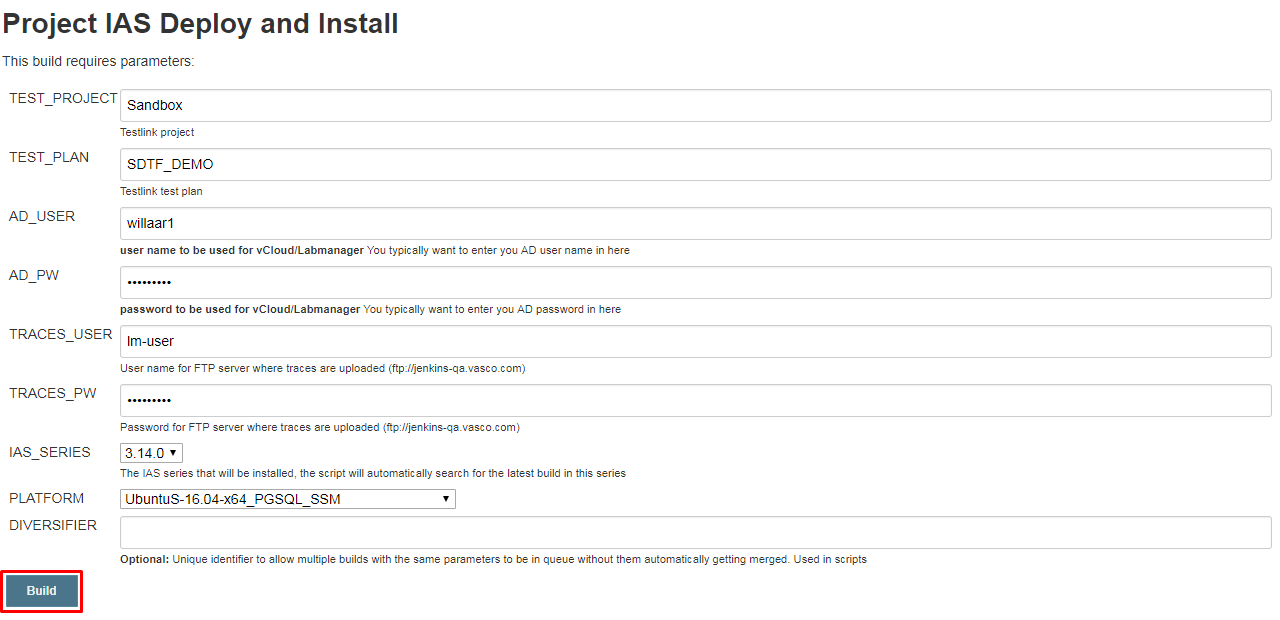
TRACES\_USER is lm-user

TRACES\_PW is Shared1234 (does this need to be redacted in my final hand in?>

IAS\_SERIES is 3.14.0

PLATFORM is UbuntuS-16.04-x64\_PGSQL\_SSM

DIVERSIFIER is optional and left empty



Click build when done

Wait for it to build, and then you’re done.

Don’t forget to give the sdtf the right platform as a startup parameter

python ~/sdtf/suites/ias/webadmin/suite.py "3.14.0.4791" "UbuntuS-16.04-x64\_PGSQL\_SSM" ChromeWebadminTestSuite --projectName="Identikey Server" --testplanName="IAS 3.14.0 - Automated Webadmin" --virt.user="lm-auto-wemmel" --virt.password=$LMPW --traces.user="lm-user" --traces.password=Shared1234 --install.media="//10.132.0.242/wqa/QC-Projects/01-Identikey/3.15.0/builds/3.15.0.2262/ias-dev\_3.15.0.2262.iso" --deployed\_name="3.14.0.4791\_UbuntuS-16.04-x64\_PGSQL\_SSM"

## Dockerfiles

### For use with the SSH tunnel

FROM python:2.7

ADD python\_test.py /

# Selectively add parts of SDTF needed for IAS testing

ADD sdtf /root/sdtf

# RUN mkdir -p /root/sdtf/suites/ias/

# ADD sdtf/suites/ias /root/sdtf/suites/ias/

# The sdtf directory may be a FUSE sshfs mount from a developer's Windows

# PC and as such, already contain files produced by instal\_sdtf{,\_linux}.py

# To avoid conflicts with the future invocation of install\_sdtf\_linux.py,

# remove them:

# RUN ls -l /root/sdtf > /root/sdtf-contents

# RUN rm -rf /root/sdtf/runtime /root/sdtf/sdtf\*activat\*

# The following command removes the carriage return line endings that cause problems when copying the file from a windows machine

# This was made necessary because we currently copy the sdtf folder via an ssh tunnel.

The above comments were made obsolete by the .dockerignore file

The following two commands are necessary to fix the line endings of the shell script

RUN tr -d "\r" < /root/sdtf/install\_sdtf\_linux.sh > /root/sdtf/install\_sdtf\_linux.sh.new

RUN mv /root/sdtf/install\_sdtf\_linux.sh.new /root/sdtf/install\_sdtf\_linux.sh

RUN apt-get update && apt-get install -y \

python-tk \

python-pip \

python-ldap \

&& rm -rf /var/lib/apt/lists/\*

RUN apt-get update --fix-missing

RUN pip install virtualenv

RUN chmod +x /root/sdtf/install\_sdtf\_linux.sh

RUN /root/sdtf/install\_sdtf\_linux.sh

RUN tr -d "\r" < /root/sdtf/sdtf-activate > /root/sdtf/sdtf-activate.new

RUN mv /root/sdtf/sdtf-activate.new /root/sdtf/sdtf-activate

ENV LMPW=U89yMN8R8bwgOKR1k6tz

CMD [ "/bin/bash" ]