[Document title]

[Document subtitle]

VASCO Data Security

[Course title]

# Preface

# Summary

# Table of contents

[Preface 1](#_Toc512841859)

[Summary 1](#_Toc512841860)

[Table of contents 2](#_Toc512841861)

[List of figures, graphs 3](#_Toc512841862)

[Introduction 4](#_Toc512841863)

[Problem statement 5](#_Toc512841864)

[Current situation 5](#_Toc512841865)

[What could be improved 5](#_Toc512841866)

[Goals 6](#_Toc512841867)

[Methodology (practical goals) 6](#_Toc512841868)

[Research (documentation, discussion and quantification of the gains) 6](#_Toc512841869)

[Technologies 7](#_Toc512841870)

[Selenium 7](#_Toc512841871)

[Selenium grid 7](#_Toc512841872)

[Docker 7](#_Toc512841873)

[What is a container? 7](#_Toc512841874)

[Why containers? 8](#_Toc512841875)

[Implementation 10](#_Toc512841876)

[Proposed architecture 10](#_Toc512841877)

[Changes needed 10](#_Toc512841878)

[Method 11](#_Toc512841879)

[Setup 11](#_Toc512841880)

[IAS on vCloud 11](#_Toc512841881)

[The containers 12](#_Toc512841882)

[Container with SDTF 12](#_Toc512841883)

[Selenium Grid 12](#_Toc512841884)

[Challenges 13](#_Toc512841885)

[IE in a container? 13](#_Toc512841886)

[Issues with the IE node 13](#_Toc512841887)

[Performance difference 14](#_Toc512841888)

[Conclusion 15](#_Toc512841889)

# List of figures, graphs

# Introduction

Introduce Vasco

# Problem statement

## Current situation

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

## What could be improved

# Goals

## Methodology (practical goals)

## Research (documentation, discussion and quantification of the gains)

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.

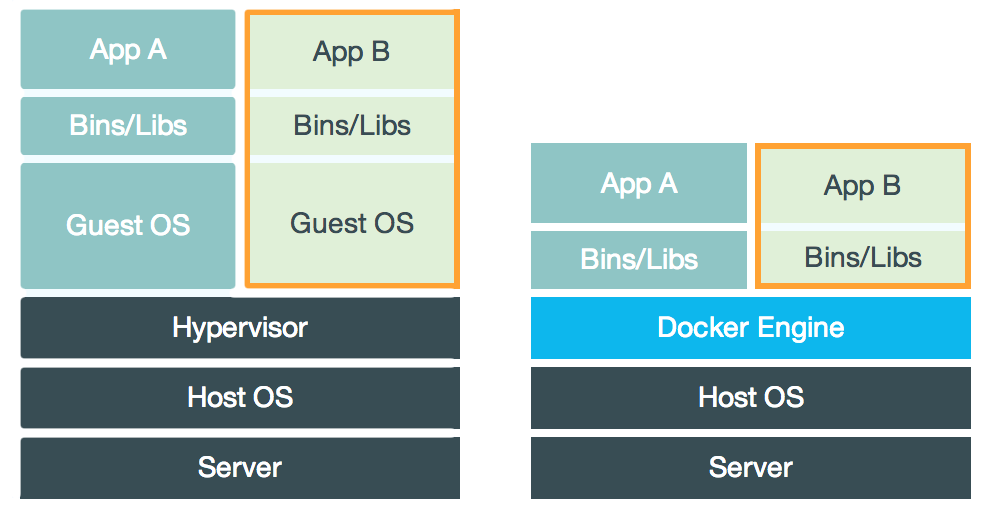
In this specific test suite, we make use of Selenium Webdriver. This service accepts commands, sends them to the relevant browser through a browser-specific browser driver, and then retrieves the results. Most browser drivers launch and access the browser application directly.

### Selenium grid

## 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 would be an actual, real computer, instead of just a simulation. A VM runs on a physical machine 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?

#### Conclusion

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

# Method

Walkthrough of the steps taken to accomplish the research goal

## Setup

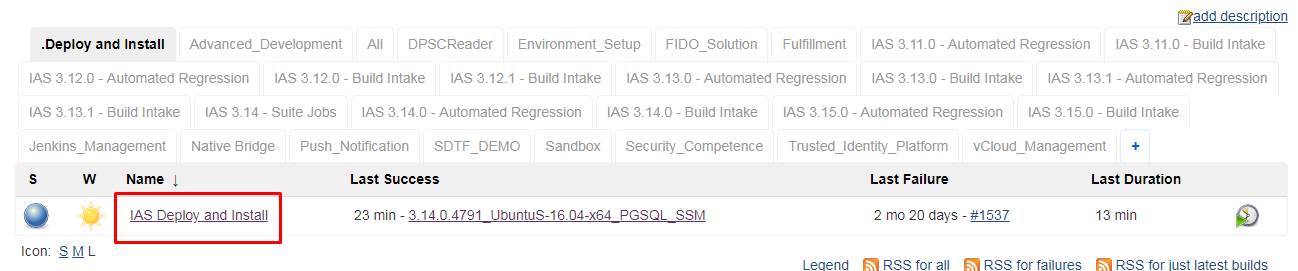
## IAS on vCloud

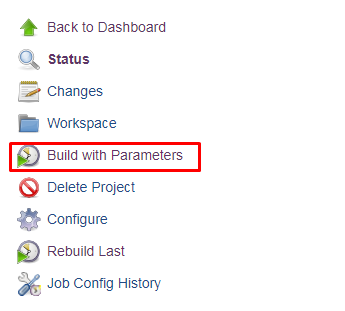
This step describes how to set up an Identikey Administration

<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

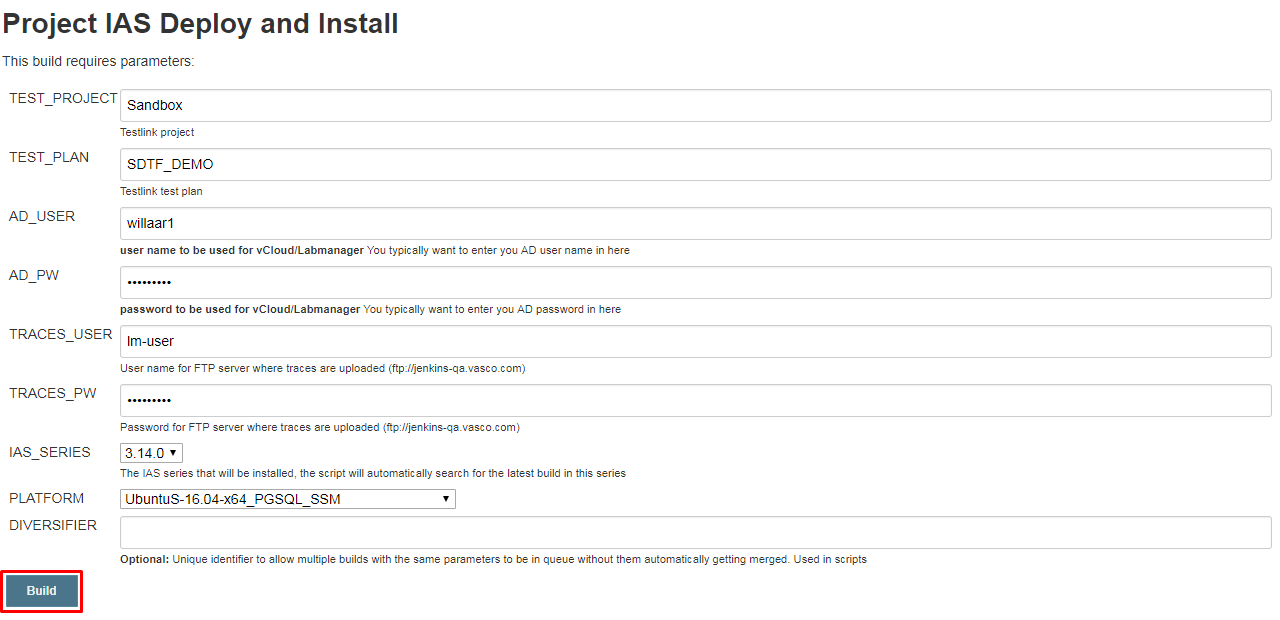
TRACES\_USER is lm-user

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"

## The containers

### Container with SDTF

### Selenium Grid

# 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!!!