# From Bare Metal to Private Cloud: Introducing DevSecOps and Cloud Technologies to Naval Systems

by

Robert Anderson

A thesis submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Master of Science

Auburn, Alabama
August 5, 2018

Keywords: Cloud Computing, Security, DevSecOps, Linux Containers, Virtual Machines

## Abstract

Virtualizing computing resources provides a myriad of benefits ranging from increased hardware utilization, better disaster recovery, and isolation between the applications and the underlying hardware. Isolation of applications through virtualization presents a significant security increase from the traditional bare metal deployment model and allows for infected machines to be easily deleted, recovered, and reprovisioned to maintain high uptime of mission critical systems. For military and government entities, these abilities are too promising to ignore and motivate many to make the transition to utilize virtual machines in their daily operations. However, handling virtual machines at scale requires much more than hypervisor technology. To fully reap the benefits of virtualization, it is necessary for companies to transition to a scalable virtual infrastructure. This makes cloud technologies and the private cloud deployment model a highly attractive solution because of its provisioning capabilities and allows consumers to maintain tight control over their physical and data security. Incorporating this infrastructure presents a daunting task and the complexity around these technologies creates confusion that act as a stumbling block to potential adopters. To make an informed transition requires careful research into state of the art virtualization technologies, processes that put security to the forefront such as DevSecOps style methodologies, and monitoring systems for enhanced observation capabilities. When brought to life, these technologies present a roadway to reliable and secure virtualization. To prove this, research was performed with a defense contractor that desired to make a transition from bare metal deployment to a more secure and scalable solution. We then developed a virtualization plan, provided a sophisticated monitoring solution, created our own DevSecOps plan to govern their virtual infrastructure, and demonstrated a proof of concept private cloud to serve as a model for their own operations.

Acknowledgements

I would like to thank my advisor Dr. David Umphress for his support and guidance that were critical to performing the work in this thesis. His advice and planning have been immensely helpful to completing my master's degree at Auburn University. I would like to thank my committee members Dr. James Cross and Dr. Dean Hendrix who have supported me from the beginning of my college career and for teaching me the foundational aspects of software engineering. I would like to thank the company Progeny along with Scott Lewis and Jamie Griffith. Their consultation and feedback as our clients have been a wonderful experience. I would also like to thank Rodrigo Sardinas whose expertise and work in cloud computing was invaluable to my thesis.

Table of Contents

## List of Tables

## List of Illustrations

## List of Abbreviations

| | |
|---|---|
| Cgroups | Control Groups |
| CPU | Central Processing Unit |
| GPU | Graphics Processing Unit |
| GUI | Graphical User Interface |
| IDS | Intrusion Detection System |
| IO | Input/Output |
| IPC | Interprocess Communication |
| KVM | Kernel-based Virtual Machine |
| MAC | Mandatory Access Control |
| NIST | National Institute of Standards and Technology |
| NUMA | Non-uniform Memory Access |
| OS | Operating System |
| Seccomp | Secure Computing |

# CHAPTER 1 – Problem Description

We were asked by a research sponsor to investigate how security can be enhanced by transitioning from a bare metal deployment model to one using virtualization. Our customer outlined that:

- Virtual machines provide a barrier and needed isolation between applications and hardware.
- Virtual machines are easier to replace in comparison to attempting to clean infected servers.
- Recovery and snapshot capabilities make system recoveries easier.
- Virtual machines provide better partitioning and utilization of resources.

## 1. Requirements

At the beginning of the project, our customer brought two main requirements: to recommend the most promising state of the art virtualization technologies for the expansion of their virtual infrastructure and to provide a general monitoring solution with the ability to provide detailed information of applications running within a virtual environment. With the first requirement, it was necessary to provide a virtualization plan that would fully inform our customer of the different choices of virtualization technologies that would prove the most beneficial to its operations. This included hypervisor choice and the exploration of emerging virtualization technologies such as Linux containers (operating system virtualization). For monitoring software, our customer required a generalized solution that could work with a variety of different virtual machines. Along with this, it was necessary that the monitoring solution provided a variety of data (CPU, Memory, IO) and at a detailed level that would provide enough information to make an informed decision on the behavior of the application or run through a rule-based intrusion detection system. With use of virtual machines there needed to be a comprehensive management solution that allowed for governance and allocation of virtual resources. Along with this arose a need for a process to deploy software on their virtual infrastructure to provide the highest degree of security and a necessary standard for deployment. Finally, consultation with our customer revealed that false positives from alert systems can lead to personnel constantly ignoring messages from critical systems and even when observed, problems with applications are hard to diagnose. This allows for application vulnerabilities to be exploited and for malicious use of resources. Therefore, a solution utilizing artificial intelligence to understand monitoring information and to determine when an application is behaving maliciously exemplified great business value. This left us with a final list of requirements:

- Develop and relay knowledge on virtualization technologies with an emphasis on security.
- Develop a general monitoring solution for virtual machines.
- Develop a solution to handle virtual machines at scale.
- Devise a deployment plan with an emphasis on security.
- Develop an intelligent agent that can recognize malicious behavior from our monitoring solution.

# CHAPTER 2 – Background

To fulfill the requirements set by our customer, knowledge was built to fully understand and formulate our solution. Our solution utilizes a wide range of virtualization, monitoring, and artificial intelligence technologies that each play a central role in creating, managing, and monitoring virtual environments.

## 1. Virtual Machines

A virtual machine has traditionally been the disk image of a standalone environment, complete with operating system and accompanying applications.  As a guest, a virtual machine should have no knowledge of its host, but the reverse is not necessarily true.  The host may have to simulate every instruction executed by the guest (complete virtualization).  It may have to simulate instructions or services deemed risky to run natively (partial virtualization).  Alternatively, the guest may have to be modified to replace low-level service requests with calls to the host's operating system (paravirtualization).  In all cases, the guest runs in its own protected address space and is isolated from the rest of the physical system. A virtual machine achieves this isolation by utilizing a hypervisor.

1) Hypervisor

A hypervisor is software that isolates and makes a virtual machine independent from the underlying host system. This allows the virtual machine's operating system to be different from its host (e.g. Linux virtual machine running on Windows). Hypervisors are divided into two different types. Type 1 hypervisors (bare-metal hypervisors) operate directly on the host's hardware. Type 2 hypervisors (host hypervisor) requires running on top of a host operating system.

## 2. Linux Containers

Container-based virtualization is much different than traditional methods such as hypervisor-based virtualization. Hypervisor-based virtualization requires the entire operating system to be virtualized for each guest. This overhead makes it harder to scale with current machinery and can possibly provide more isolation than needed. Container-based virtualization solves this by containers sharing the same host operating system. This makes container-based virtualization more resource efficient. To perform this virtualization, containers use kernel features such as namespaces and control groups to make virtual environments at the operating system level. Container platforms (e.g. Docker) use these features to generate containers and isolate applications from the host machine and other containers [Xavier, 2013].

1) Namespaces

Namespaces provide virtualization between the operating system and the container. This means from the container's perspective, it does not have knowledge of host system resources such as process IDs, mount points, etc. The container can then have its own process ID space and user IDs (such as 1 reserved for *init*) that are already in use on the host system. Listed below are some the existing namespaces [Man Namespaces]:

- PID – virtualizes process IDs.
- Network – virtualizes network devices, stacks, ports, etc.
- Mount – virtualizes files system mount points creating different views of the filesystem.

- User – isolate security-related identifiers and attributes: user IDs, group IDs, etc.
- UTS – virtualizes the system identifiers: hostname and NIS domain name.
- IPC – virtualizes IPC resources: System V IPC object, POSIX message queues, etc.

2) Control Groups

Control groups (cgroups) provide a means of limiting computing resources that are accessible to contained applications. Cgroups follow a hierarchical structure of subsystems similar to the Linux process model which allows for children to inherit attributes of their parents. This enables fine-grain restrictions of system resources. Processes are added to these groups and are governed by the set limitations. Listed below are some of the existing subsystems [Man Cgroups]:

- cpusets: bind to specific CPUs or NUMA nodes
- blkio: limits block io
- cpuacct: accounting of CPU usage
- devices: control which tasks may create, create, or read (mknod) devices
- freezer: can suspend all actions of processes
- hugetlb: limits the use of huge pages
- memory: limits process memory, kernel memory, and swap
- net_cls: used to place a classid on network packets created by a cgroup
- net_prio: used to prioritize network traffic for different cgroups
- cpu: used to guarantees that a cgroup will get a certain amount of CPU time
- perf_event: allows for performance monitoring of processes within the cgroup

3) Docker

Docker is a container platform that utilizes Linux kernel features to provision containers and has emerged as the industry standard for container technology. To uncover how to introspect containers we inspected the Docker architecture and some of the lower level workings of Docker.

1. *Docker Architecture*

Docker has a relatively simple architecture. See the illustration below.

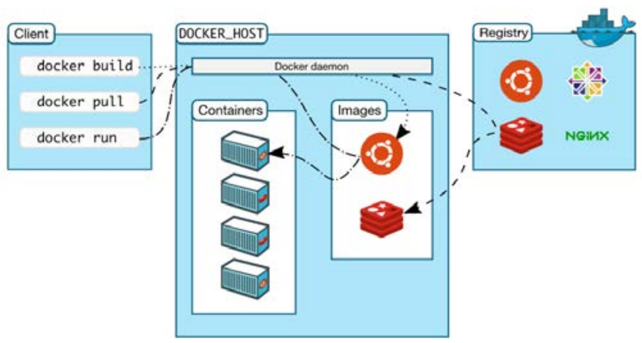


Illustration 1 - Docker Architecture [Docker, 2017]

Docker uses a client-server architecture in which the Docker client communicates with the Docker daemon on the host machine. The Docker daemon creates containers based on a Docker image that is stored in a registry [Docker, 2017].

*a) Docker Image*

A Docker image serves as the template for the creation of all Docker containers. Images are created by building a Docker file resulting in a Docker image. Docker files are a series of instructions that download software, setup the container environment, and start any necessary applications. The Docker daemon builds the Docker file by running each of the instructions one-by-one and ending with a Docker image. This image can then be stored in a registry for future use.

*b) Docker Client*

The Docker client serves solely as an interface to communicate to the Docker daemon. The Docker client can be located on the host machine or can be on a different computer to give commands from another machine. Users enter commands through a provided command line utility by using the “docker” command. Commands are then taken and sent to the Docker daemon.

### *c) Docker Daemon*

The Docker daemon does most of the work in the Docker architecture. Its responsibility is to listen for Docker commands from the client and execute them on the host machine. The Docker daemon is responsible for many critical features including container creation, container management and deletion, pulling images, etc.

### *d) Docker Registry*

A Docker registry is a repository of Docker images. The Docker daemon pulls images from a registry to create containers. A Docker registry can be local or outsourced to the cloud. An example of a registry is Docker Hub which is provided by Docker. Docker Hub has become a popular repository and allows people to use and share images.

## 2. *Union File System*

Docker uses a union file system to manage files for each container. Each Docker image is divided into layers. These layers are created when individual commands in the Docker file are executed and are joined together to form one image. The union file system allows containers to use the files of each layer and enables copy-on-write (COW) capabilities. This makes containers that share a common image provide greater memory efficiency.

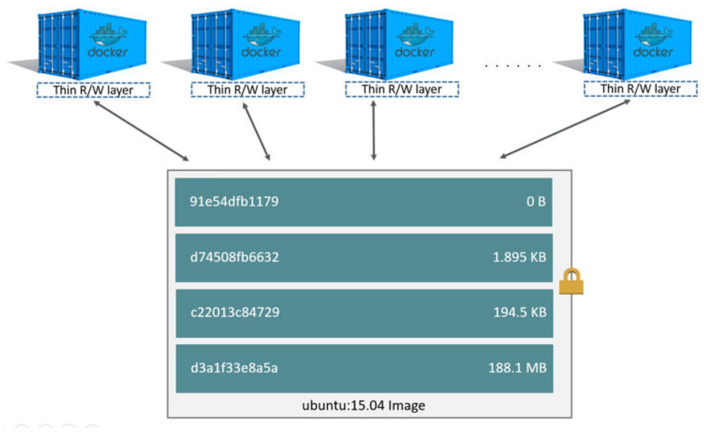


Illustration 2 – Image Layers and Containers [Docker, 2017b]

Files are shared by each container and are read-only until a file is altered. If a contained application writes to a file, the container that wrote to the file receives its own copy. The other containers' copy is left untouched and remains shared until another needs to perform a write operation [Hines, 2015][Pahl, 2015].

## 3. Monitoring Applications and Techniques

Monitoring was one of the highest priority requirements of our research and significant effort was given to establish a cohesive solution that would provide detailed monitoring information.

### 1) Virtual Machine and Container Introspection

Introspection is the analysis of virtual environments from the outside to harness information of software running in virtual environments. Introspection reveals the state of the guest machine to better understand what operations are taking place and to detect any erroneous or malicious behavior of the system. Introspection works to solve a central security problem when monitoring virtual environments. When setting up virtual infrastructure, system administrators must compromise between placing intrusion detection system (IDS) on the virtual machine to have better vision into the virtual environment or attempt to monitor from the outside where it is safely deployed from any malicious applications. Introspection allows for the best of both situations by peering into the hypervisor from the outside and better understanding the state of the virtual machine [Garfinkel, 2003].

## 4. Cloud Computing

According to NIST (National Institute of Standards and Technology), “Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources that can be rapidly provisioned and released with minimal management effort or service provider interaction. [Mell, 2011]” Cloud computing is the main paradigm for building and outsourcing virtual infrastructure. It has become popular because it allows adopters to take advantage of economies of scale that allow for cheaper and better utilization of hardware. Along with this, it’s able to free companies from having to create and manage their own virtual infrastructure solution. Clouds come in many forms and vary based on how much companies are willing to expand their trust boundary. There are four main models for cloud deployment: private cloud, community cloud, public cloud, hybrid cloud. Private clouds are for the use of one organization and requires little to no trust in another organization to manage the cloud. Community clouds serve a community of organizations that have similar motivations. Public clouds are open to the public and require the highest level of trust in a third party. Hybrid clouds are a mix of these different models to provide a middle ground that best suits a company’s ability to compromise between efficiency and security.

### 1) Openstack

Clouds are deployed using software such as Openstack. Openstack is an open source cloud operating system used to provision and manage virtual resources. Openstack is divided into many projects to achieve cloud functionality. Arguably the most critical project of Openstack is Nova which is used for all compute services.  This makes Nova responsible for creating, deleting, and managing virtual machines. To enable networking, Neutron is the main networking service of Openstack and allows for the creation of virtual networks. Openstack contains support for object and block storage through Swift and Cinder along with virtual image storage through Glance. These examples represent only the most core and essential software of the Openstack project. The capabilities of the current Openstack projects span from dashboards (Horizon) to bare metal container deployment (Ironic) and provide rich functionality for a diverse set of use cases.

## 5. Deep Learning

To increase our detection capabilities, utilizing artificial intelligence and deep learning was thought to be the most promising area. Deep learning is a subset of machine learning and is centered around the use of neural networks. Deep learning attempts to mimic the biology of living creatures and create networks of neurons that are used to classify and extract features from data. These networks are trainable by collecting or generating training data and using labels (supervised learning) to indicate a classification that will allow the network to make judgements on future data samples.

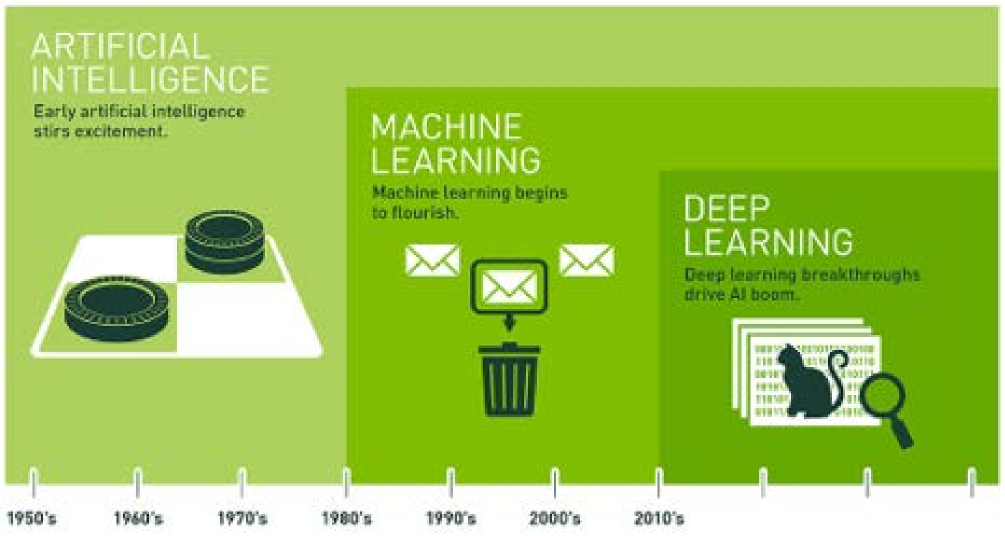

ARTIFICIAL INTELLIGENCE
Early artificial intelligence stirs excitement.
MACHINE LEARNING
Machine learning begins to flourish.
DEEP LEARNING
Deep learning breakthroughs drive AI boom.
1950's
1960's
1970's
1980's
1990's
2000's
2010's

Illustration 3 – Subsets of Artificial Intelligence Over Time [NVIDIA, 2018]

### 1) Deep Learning Libraries

To speed up development of our intelligent agent, we utilized several libraries centered around the creation of neural networks. Tensorflow is an open source software library that is targeted towards creating deep learning applications and utilizes GPUs to speed up computations that are critical to training and predictions. Tensorflow provides features such as premade neural network layers, optimizers, and loss functions that can immediately be added to a neural network. Although Tensorflow provides great functionality to developers, it can still be difficult to use. This has inspired the creation of higher level APIs that further simplify neural network creation and use deep learning libraries as a backend. Keras is a high-level API that has wide support through online tutorials and its model building greatly simplifies the creation of neural networks. Adding layers to a sequential neural network in Keras follows a building block mentality where each layer can be added to the model one by one. Along with simple network development, Keras provides wide support for a multitude of different networks. Below is an example of adding layers to a Keras sequential model.

```
def buildModel(sampleSize, numberOfFeatures):
    model = Sequential();
    model.add(LSTM(32, input_shape=(sampleSize, numberOfFeatures)));
    model.add(Dense(2, activation='sigmoid'));
    return model;
```

Illustration 4 – Building a Model in Keras

# CHAPTER 3 – Solution

## 1. Overview

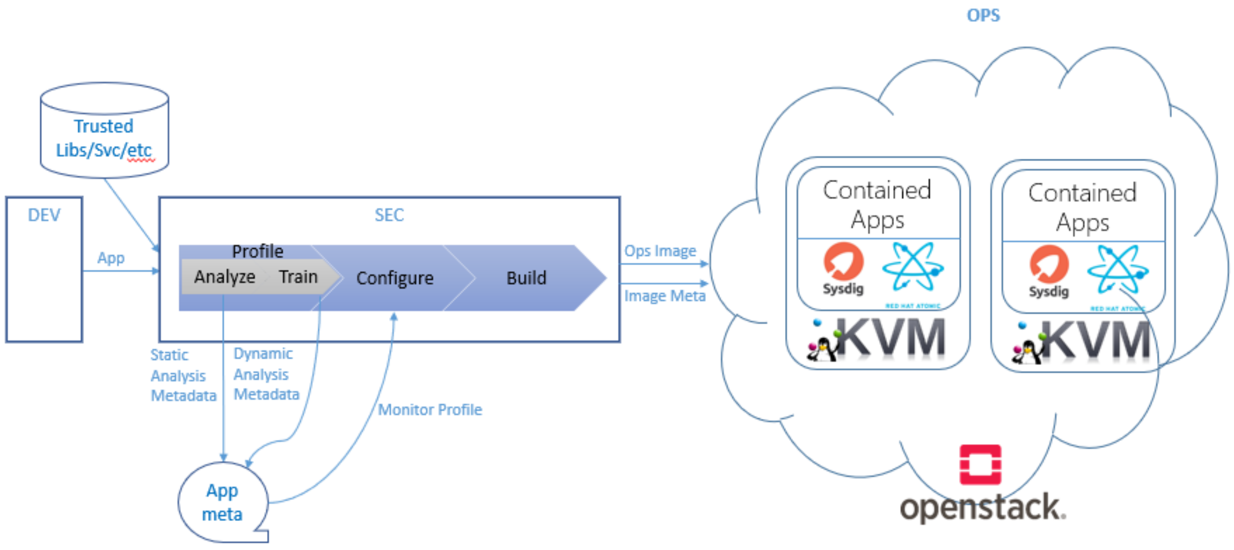


Illustration 5 – Overall Cloud Solution

The solution above is centered around providing as much isolation as possible and strengthening the security of virtualization. Starting with the choice of virtualization technology, the solution utilizes containers in conjunction with virtual machines. The use of containers gives us significant benefits with little performance costs and provides an extra layer to defend within the virtual machine. For our container platform, we chose Docker as it is the industry standard and has taken center stage over other alternatives such as LXC. For our choice of virtual machine, we decided to use a minimal container operating system called Atomic Host. This new type of operating system puts containers center stage and gives a first-class treatment to container security. These virtual machines and containers are then managed by a private cloud deployed at Auburn University, and the applications running within them are monitored using the open source kernel module Sysdig which provides granular and detailed information to make informed decision making by intrusion detections systems. The operations and flow of control from development to cloud is guided by our DevSecOps process that provides a clear path to deployment on our private cloud and gives a high emphasis on increasing security, particularly where containers and Linux container features contain vulnerabilities. Work has also been expended on using machine learning and the creation of an intelligent agent to take information from our monitoring agent and utilize it to detect anomalous behavior, but this is currently a work in progress.

## 2. Virtualization Plan

One of the most fundamental questions of this research was what virtualization technologies to utilize. This choice would form the foundation of how other pieces of the solution would work together to accentuate the isolation capabilities of our virtual infrastructure and work to mitigate its weaknesses. Containers proved to be a main driving force when creating our virtualization plan. The capabilities of containers to isolate applications at an operating system level were highly attractive and were viewed as a new level of granularity to provide isolation and security benefits.

1) Linux Containers

Containers are currently undergoing a rise in popularity within the software industry. Currently, Docker boasts over twenty-nine billion container downloads and over nine hundred thousand applications in Docker Hub (their docker image storage solution) [Docker, 2017c]. This surge in use represents that the technology could provide significant business value to our customer and as a virtualization technology, provide greater application isolation.

At first glance, container virtualization can often be perceived as "virtualization on the cheap." However, container virtualization presents a new level of granularity of virtualization technologies. Containers work at a different level than traditional virtual machines.

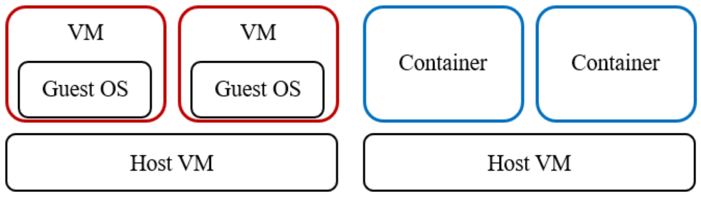


Illustration 6 - Virtual Machines vs. Linux Containers

Looking at the illustration above, we can see the main difference between virtual machines and containers. Virtual machines virtualize the entire operating system while containers share an operating system. From this difference stems most of the benefits and drawbacks of container virtualization.

1. *Container Benefits*

The primary benefit of containers is that they provide an additional layer of isolation. This gives developers another virtualization option for software deployment. Containers isolate themselves by using features in the operating system (in our case Linux). For example, the namespace feature in Linux provides containers the ability to have their own resources such as their own file system, network devices, users, etc. Another feature called cgroups ensures that one container does not use too many resources ("noisy neighbor problem") and starve another contained application. Because of this increased isolation, containers grant more control over the environment of individual applications. Each application gets its own environment (file system, process list, etc.) where one application cannot affect the environment of the others. So instead of the traditional model of software applications sharing a server or virtual machine, each application has its own contained environment within that machine. This makes it much more difficult for applications on the same machine to affect the state of another and withhold resources that another application may need. This may concern some developers because certain applications may need certain devices on the host operating system and require communication with other applications on the same machine. However, Docker gives a lot of freedom to developers to control communication between containers and allows for developers to expose certain devices to containers that require them.

Increased isolation often comes at the cost of memory and CPU efficiency, but containers differ in that they are highly efficient with both. As mentioned previously, containers do not virtualize the entire operating system, but instead, share it with other containers. In addition to this, containers use a copy on write file system (union file system) that ensures that containers only receive their own copy of a file when modified by activities within the container. Until then, files for the container are read only. This saves space on the system when containers are deployed on large scale enterprise systems.

Containers virtualization has very little efficiency impact associated with increased isolation. Containers work near natively because of very little software coming between contained applications and the operating system. [Felter, 2015][Joy, 2015]

Containers can be deployed and redeployed rapidly thanks to docker images being built beforehand and awaiting near instant deployment. This gave rise to the idea of “immutable infrastructure” [RedHat] that is designed to help create easy and clean deployment of applications. In immutable infrastructure, containers are not updated but are completely refreshed. Containers are easy and quick enough to deploy that updating a container is a more arduous and time-wasting task that is prone to error. Therefore, creating a container with the newest update and replacing the older container has become a better option.

### 2. *Container Drawbacks*

Although there are numerous benefits to container virtualization, there are still drawbacks. One of the primary drawbacks is how much isolation that containers provide. Through this research, we believe containers provide significant isolation benefits, but containers do not perfectly contain. This relates back to containers sharing the same operating system. Any resources that are not virtualized and are shared between containers are not contained. The Linux kernel token ring is an example of this [Walsh, 2014]. As these vulnerabilities make themselves known they often can be mitigated or eliminated.

Another problem with containers is that Docker containers are more secure by default but can be configured to the point of an application running uncontained (privileged containers). Docker allows consumers to grant as many capabilities and resources as they desire for applications to function within their environment. Some examples of these applications are kernel modules and programmable logic controllers that require more operating system resources. This allows those using Docker to remove all the safeguards present in containers and requires discipline during software deployment to achieve the highest degree of isolation.

Some potential adopters may be slow to use Linux containers because of its regard as a new form of virtualization, but container technology has existed much earlier than Docker's rise to fame. The features utilized by Docker containers have existed for quite some time in the Linux kernel and previous forms of containers have existed before Docker (e.g. Solaris zones, jails, etc.) [Price, 2004].

### 3. *Container Vulnerability Mitigation*

Many of the vulnerabilities in container virtualization can be mitigated to avoid flaws in virtualization and prevent container breakouts. Most of these techniques are centered around ensuring contained applications never have access to a resource that could affect the state of another container.

#### *a) Linux Kernel Features*

The Linux kernel already supports multiple features that help segregate container environments. Some are integral and built-in to container security (cgroups, namespaces) while others can be added and configured to provide greater security (seccomp, capabilities). Taking advantage of kernel features provides a highly generalized solution that applies to all Linux machines. This makes using these features highly promising to improve isolation and security for all Linux applications.

### *b) Secure Computing (seccomp)*

Secure computing (seccomp) is a Linux kernel feature that is designed to eliminate system calls for a contained application. This allows software deployers to reduce the interface of the Linux kernel through software and tailor it to each deployed application. Seccomp utilizes a whitelist and allows for deployers to only list what system calls are needed for a specific application. This does several things. It bars applications access to knowingly flawed and obsolete Linux features and helps ensure that applications only use the system calls needed to operate. For example, if an application was redirected to execute malicious code that could not work within the limitations of the listed system calls, it would be unable to fully carry out its payload. This protects the system and can make attacks either impossible or require a higher degree of sophistication. When using Docker containers, a seccomp profile is added by default. However, it is very loose and blocks only obscure and rarely used system calls. Therefore, it is up to those who develop and deploy the application to create their own tailored profile to assure the highest security gain possible [Chandramouli, 2017][Man Seccomp].

### *c) Linux Capabilities*

Linux capabilities divide the permissions of superuser and allow for permissions to be granted piecewise as needed. This is different than running applications as privileged and unprivileged because privileged applications no longer have to run with many unnecessary permissions. The list of capabilities is quite long and will not be added to this paper but are available in the Linux man pages. However, the capability “CAP_SYS_ADMIN” is particularly powerful because it has been a catch-all for many permissions and should not be given lightly [Man Capabilities] [Lopez, 2015].

### *d) Mandatory Access Control (MAC)*

Mandatory access control attempts to enforce security policies on system resources to forbid malicious behavior. Mandatory access control systems (e.g. SELinux and AppArmor) are fully compatible with

Docker containers and provide another mechanism to forbid and restrict resources that would otherwise go uncontained. Mandatory access controls can express a much greater control in preventing breakouts and strengthening container virtualization. When using SELinux (security-enhanced linux), we can forbid specific resources that are not namespaced and are not virtualized at the operating system level. This allows us to fill the holes in container isolation. Along with remedying specific instances, SELinux has strong defaults. When SELinux does not have a policy for a resource, any request results in denial. This makes unforeseen vulnerabilities less likely and prevents them before their discovery.

2) Virtual Machine

The desire to mitigate vulnerabilities of containers inspired research into a virtual machine that would provide isolation by virtualizing the operating system and strengthening the containers residing within its environment. Minimal container operating systems virtualize the operating system while putting containers to the forefront.

### 1. *Container Operating Systems*

Because the kernel is shared between containers, one of the best means is to use an operating system that is designed specifically for containers and treats them as the primary means of running software applications. They are much better at managing contained applications and are recommended over general-purpose operating systems [Souppaya, 2017]. Some examples of these are Container Linux by CoreOS (recently acquired by Red Hat), RancherOS, and Red Hat Atomic Host.

Container operating systems are typically minimal in size and attempt to eliminate unnecessary software by not including it by default. This increases scalability and is well suited to running within a cloud environment. Additionally, this reduces the attack surface of the operating system by eliminating unneeded services that could be exposed to the network and serve to hinder critical operations.

Container operating systems present a paradigm change when compared to traditional operating systems [RedHat, 2017]. Container virtualization’s near native CPU performance and efficient file system has urged OS developers to force the use of containers. Therefore, every application runs contained on a container operating system. This has incorporated the idea of immutable infrastructure and “atomic” operations into the operating system. This makes deployment with these operating systems a much cleaner process by eliminating mistakes associated when configuring applications and application environments that can result in failed deployment and significant security risk to deployment systems.

## 2. *Cloud Hypervisor*

When deploying a private cloud, a hypervisor is needed for the compute service to provision virtual machines. Openstack supports multiple different hypervisors, but ultimately, we decided to choose KVM (Kernel-based Virtual Machine). KVM provides an open source solution that meets our hypervisor needs. Because KVM is a part of Linux, KVM has a wide user base and large support from the IT community. This has made tool support for KVM strong from both public domain and commercial sources. Along with this, KVM better supports introspection capabilities of running applications. This keeps the avenue open of performing introspection at the compute level of our Openstack cloud.

## 3) Final Virtualization Plan

After researching different forms of virtualization and related technologies, we settled on using Docker containers within the minimal container operating system Red Hat Atomic Host. All of this is run on our compute node of our Openstack cloud that utilizes KVM as the hypervisor.

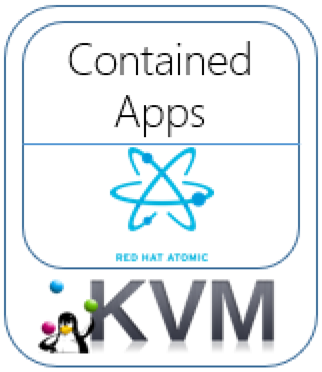


Illustration 7 – Virtualization Technologies Working Together

When trying to use container virtualization, it is a misconception that one cannot mix different forms of virtualization, but virtual machines and containers complement each other and improve upon a pure container or pure virtual machine approach. With a pure container approach (e.g. Openstack Ironic), containers do not provide the same degree of isolation as virtual machines but are much more scalable and introduce very little overhead. Pure virtual machines provided needed isolation but creating a virtual environment for each application is costly. Therefore, containers fill this void and provide more isolation where using only virtual machines is not a cost-efficient option.

## 3. Monitoring

Monitoring is used widely in industry and provides the ability for system administrators and software applications to detect malicious or erroneous behavior occurring within computer systems. For our monitoring solution, our focus was the ability to perform introspection and the ability to introspect containers. Monitoring from outside the containers gives us the benefit of having a layer of virtualization between our monitoring solution, shielding it from tampering of misbehaving applications. It was also necessary that our monitoring solution provide detailed monitoring information to make the most informed decisions possible. A kernel module named Sysdig met these requirements.

### 1) Sysdig

Sysdig is an open source kernel module that gives us the ability to monitor containers from the outside. Along with this functionality, Sysdig provides an extensive and diverse range of information covering CPU, memory, and network operations. Sysdig also provides monitoring data at a very granular level allowing us to see individual system calls, active file handles, and network information of contained applications. Below is a small sampling of Sysdig's monitoring capabilities.

| CPU | Memory | Network |
|---|---|---|
| - see top processes in terms of CPU usage. | - see all open file handles | - top processes by network usage |

| | | |
|---|---|---|
| - number of threads and children of process<br>- see all system calls of process (including arguments)<br>- see top system calls where most time has been spent<br>- see top system calls returning errors<br>- number of cgroups a thread belongs to | - observe IO activity on file by name<br>- show every file open that occurs in a directory<br>- page faults by process<br>- amount of virtual memory for main thread of process<br>- top processes and files by IO errors<br>- see all failed disk IO calls | - network data exchanged with IP<br>- top local server ports in terms of bytes and connections<br>- top client IPs in terms of bytes and connections<br>- see all GET HTTP requests<br>- see all SQL select queries made by a machine |

Table 1 – Example Sysdig Capabilities for Monitoring Information

With this amount of information, robust filtering capabilities are needed to parse through this data. Sysdig provides a filtering solution that allows you to mix and match filters to narrow down a search space to extract the information needed. This can be performed by using the Sysdig command line tool, a graphic user interface (csysdig), scripts (Sysdig chisels), and APIs. This enables programmatic solutions to be able to make informed decisions on information gathered using Sysdig. This spurred the creation of an open source intrusion detection system called Sysdig Falco that alerts on Sysdig monitoring information using a rule-based approach. Although Sysdig Falco was tempting to use for the purposes of securing containers, a rule-based approach seemed rigid and a source for false positives.

2) Intelligent Detection System

False positives and human incapability to remedy malfunctioning computer systems were problems that naturally arose out of conversations with our customer. Because of this need, we found it valuable to invest in the creation of our own intelligent detection system. Currently the solution is a work in progress, but significant progress has been made.

### 1. *Intelligent Detection Architecture*

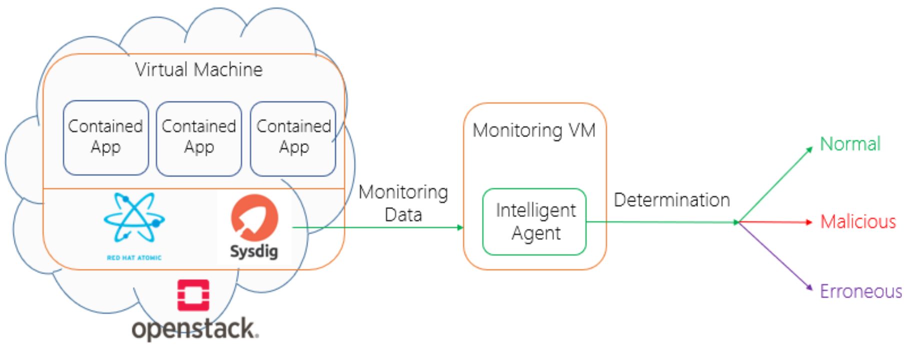


Illustration 8 – End Goal Intelligent Detection Architecture

Above we can see our proposed architecture for integrating the intelligent agent with the cloud infrastructure. All applications run within their virtual environment and are actively monitored by Sysdig. This information is converted to JSON (JavaScript Object Notation) forwarded to a virtual machine that contains our intelligent agent. The monitoring data is then parsed into a form that the intelligent agent can predict and train on. The intelligent agent then learns what CPU, memory, and IO operations are considered as normal for each application, and then tries to determine if future actions are normal, malicious, or erroneous. This information would then trigger an alert or perform necessary remedial action.

### 2. *Neural Network*

The current intelligent agent is a deep learning solution that is aimed to differentiate normal from abnormal application behavior based on system calls. This serves as a proof-of-feasibility for future efforts attempting to improve applications security through artificial intelligence. It is built in Python and uses the Tensorflow and Keras libraries to facilitate development. These libraries have allowed for much quicker experimentation and are particularly useful to those taking their first steps into developing neural networks. Our current solution is a recurrent neural network using a simple, single-layer LSTM (long-short term memory) network followed by one fully connected layer. We chose an LSTM network because of a recurrent neural network's ability to understand temporal sequences and avoid vanishing gradients. This was a critical feature because our current focus is on understanding sequences and ordering of system calls made by an application. In our experiments, we map each Linux system call to a number and compose sequences for our network to classify each as a normal or abnormal sequence of system calls. We then generate training data based on permutations of acceptable and disallowed system calls that try to accurately depict application behavior. We then train our network on a subset (80%) of these permutations. Loss is calculated using a cross entropy loss function and the Adam optimizer is used to update network weights. The rest of the permutations (20%) are used for testing to provide a final accuracy for the intelligent agent.

## 4. Private Cloud

In the early parts of our research, our customer was mainly concerned with monitoring virtual machines. As time progressed, it became obvious that there was a significant need for software to help manage their virtual infrastructure. A private cloud solution that is owned, operated, and managed by our customer gives the benefits of cloud software to provision and allocate virtual machines while allowing our customer to keep control of sensitive data and physical security. This fulfilled our requirement for a management solution and provides a multitude of benefits.

### 1) Private Cloud Benefits

Cloud computing has some essential characteristics that grant it more capabilities and higher scalability than other computing models. These include [Grance, 2010]:

- On-demand self-service - consumers of the cloud service can request the system to provision computing resources rather than through human interaction
- Broad network access - cloud resources are available over the network to many platforms (e.g. desktop, phone, tablet, etc.)
- Resource pooling - cloud resources such as CPU, memory, and network bandwidth are pooled and then divided to customer needs

- Rapid elasticity - cloud resources can scale on demand to the point of being seemingly infinite
- Measured service - cloud resource usage is quantified, managed, and made known to cloud provider

Because of these core characteristics of cloud computing, there are some intrinsic benefits to deploying your own cloud. Cloud computing provides increased computing resource availability, saving physical space, cost effectiveness, and control over computing resources [Younge, 2010].

Although cloud computing provides all these benefits there are still risks. Security provides one of the top risk factors for not adopting cloud computing [Carroll, 2011]. One of the main reasons for this hesitation is that the term cloud computing is often seen as synonymous with a publicly deployed cloud. Public clouds require an expansion of a consumer company's trust boundary to include a third party. Therefore, control over data security and physical security are now outside the consumer's control. For military and government operations, this is often infeasible. Because we have chosen the private cloud deployment model, we avoid some of the central security problems concerning cloud computing. The private cloud deployment model allows consumers to keep hardware and data on premise. This allows the cloud consumer to maintain complete control of the security of their cloud and manage their cloud how they desire.

2) Drawbacks of Private Cloud

With this power of managing your own cloud comes the responsibility of purchasing cloud equipment and some additional security concerns. As of the writing of this paper, a multinode cloud deployment requires significant hardware and presents a barrier to entry for cloud adopters. For large military operations this is not as much of a concern but can be for smaller security- focused companies.

Although we avoid many security concerns by using a private cloud, security vulnerabilities are still present. Cloud computing relies on underlying technologies that provide its capabilities. Therefore, application security, network security, virtualization security, and database security are all still very relevant with cloud computing [Hashizume, 2013]. There are also some vulnerabilities that are specific to characteristics of a cloud. Some of these include [Grobauer, 2011]:

- Unauthorized access to cloud resources - consumers need access to cloud resources and lack of permissions could be exploited.
- Shared resources and data recovery - if memory resources are not cleaned when transferring them to another cloud consumer, memory from another consumer's session could be recovered.

## 5. Deployment Process

Along with creating our private cloud, it was acknowledged that interfacing with the cloud and software deployment needed to follow a process. This process allows the incorporation of necessary security practices to help improve the strength of virtualization within the cloud and streamline deployment of applications. This made DevSecOps highly alluring because of its focus on security and its systematic handling of software artifacts.

### 1) DevSecOps Process

DevOps is a software production practice built on the ideas of lean manufacturing and agile software development. DevOps aims to help fix the disconnect between software development activities (planning, design, development, and deployment) and attempts to introduce greater collaboration and automation technologies that work to streamline the process of creating software products (e.g. continuous integration, software tracking systems, etc.) [Ebert, 2016][Fitzgerald, 2017][Cois, 2014]. After the inception of DevOps, ideas such as DevSecOps began to arise and gain popularity. DevSecOps takes the principles of DevOps and attempts to integrate better security practices [Rahman, 2016]. Because of our customer's emphasis on higher security, DevSecOps seemed to be a promising candidate to help guide our customer in deploying their software.

1. *Limitations*

Our customer is mainly concerned with activities outside of the development phase of software. The reason for this is that they are often handed software and most likely will not have influence over these activities. Therefore, our research is mostly focused on the “Sec” and “Ops” activities.

2. *Security and Operations*

Once the application binary and its associated libraries are developed, we begin profiling to gain as much understanding of the software as possible. We start by using static analysis to determine the resources needed by the software application to operate. This is critical to understand to reduce risk posed by infected applications that have too many permissions and too much access to computing resources. This is also necessary to improve container virtualization security and reduces the possibility to access of uncontained operating system resources. We then create metadata to reflect what we have learned in a usable form such as a secure computing profile, mandatory access control rules, Linux capabilities, etc.

Once we learn as much as possible from static analysis, we attempt to perform dynamic analysis to understand how the application behaves. In this step, we try to understand what normal operation of the application would be. We run the application through tests and different inputs to generate a trace of information (e.g. system calls, CPU utilization). We take theses traces and work to create an image that can be used for learning purposes in artificial intelligence and deep learning techniques. Once learned, we can make predictions using the data provided and give a better understanding if the application is behaving anomalously in comparison to its understood behavior. This area is a current work in progress for our research and is mentioned in the future works section.

Once all information possible has been gleaned from the given software and the necessary metadata is created (profiles, rules, etc.), we then begin configuring the application’s environment according to the learned information. In this phase, we take the newly learned information and begin to package and understand how to setup our virtual environment. This can result in changing necessary deployment items (e.g. Docker files, Docker command line arguments).

Once we know how to deploy our application, we then begin building any necessary items. For example, Docker files must be built to create an image for deployment and then stored in a registry for future deployment. Once this is complete and the image has been created, we transition to deploying on virtual infrastructure (Ops).

# CHAPTER 4 – Solution Validation

To prove the feasibility and demonstrate the capabilities of our research, we utilized the department's Openstack cloud to demonstrate our overall system. We decided to use an already deployed cloud for several reasons. The hardware resources needed to produce a multinode deployment of Openstack is very expensive (over $100,000). Along with this reason, deploying a private cloud on top of an already existing private cloud was viewed as redundant and would be erased as soon as the version of the underlying Openstack cloud was updated. The currently deployed cloud utilizes the desired technologies that were researched. We chose Openstack because it is a free and open source solution that is supported by multiple companies with one of its top contributors being Red Hat.

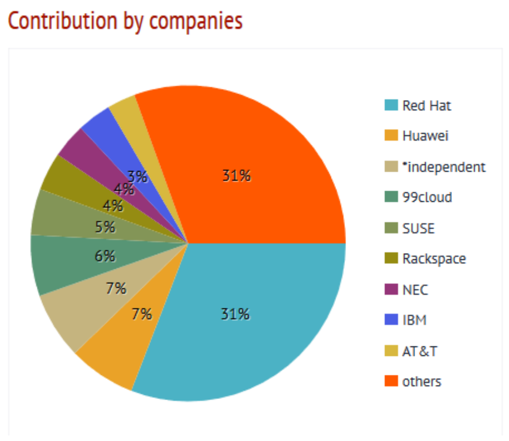


Illustration 9 – Open Source Contribution to Openstack as of June 2018 [Stackalytics]

Red Hat support was viewed as a big positive by our customer and is their standard for operations. Along with this, Kolla-Ansible was used to deploy the department's cloud. Kolla-Ansible utilizes Docker containers, Ansible, and Python to help ease and reduce the complexity of cloud deployment [Kolla]. The use of Docker containers was desirable because we wanted to utilize only one container solution, and using Ansible was seen as a positive because it does not require software on each node and utilizes ssh instead of using a unique network service. This reduces the attack surface, keeps software on nodes to a minimum, and eliminates the need for an additional network service. [Ansible].

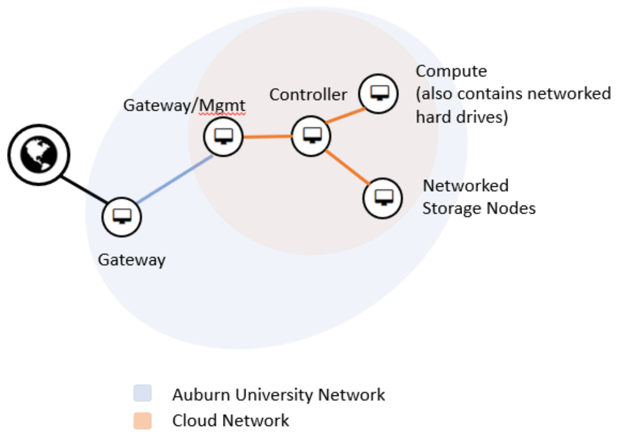
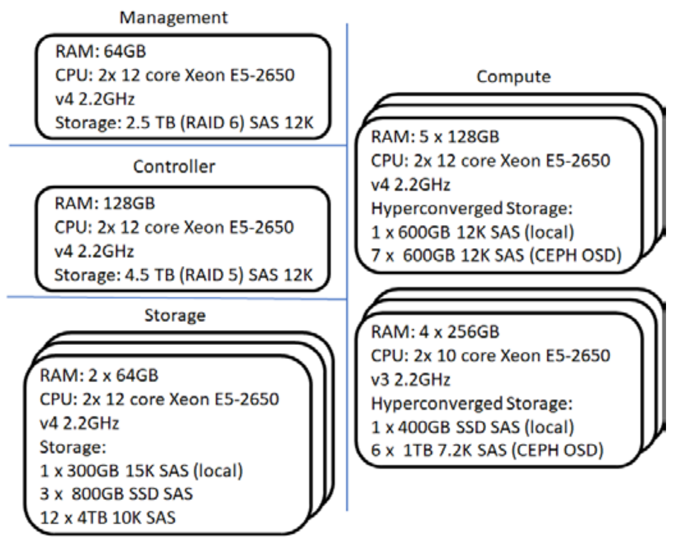


Illustration 10 - Overall Cloud Network and Architecture.

## 1. Hardware Used

Our Openstack deployment is a multinode deployment and therefore uses multiple machines. Below are the specifications of the used machines and their associated roles.



Management

RAM: 64GB
CPU: 2x 12 core Xeon E5-2650
v4 2.2GHz
Storage: 2.5 TB (RAID 6) SAS 12K

Controller

RAM: 128GB
CPU: 2x 12 core Xeon E5-2650
v4 2.2GHz
Storage: 4.5 TB (RAID 5) SAS 12K

Storage

RAM: 2 x 64GB
CPU: 2x 12 core Xeon E5-2650
v4 2.2GHz
Storage:
1 x 300GB 15K SAS (local)
3 x 800GB SSD SAS
12 x 4TB 10K SAS

Compute

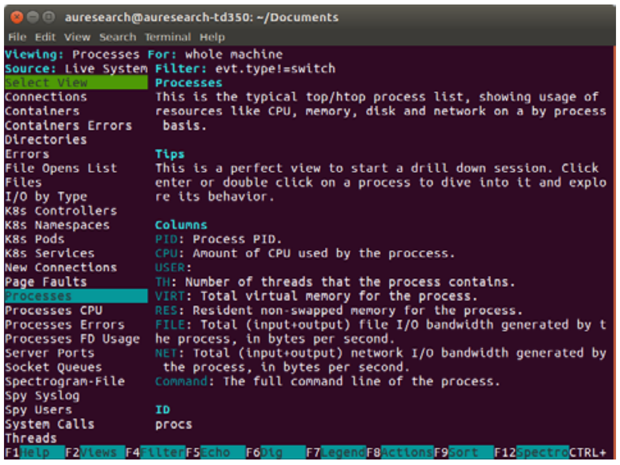
RAM: 5 x 128GB
CPU: 2x 12 core Xeon E5-2650
v4 2.2GHz
Hyperconverged Storage:
1 x 600GB 12K SAS (local)
7 x 600GB 12K SAS (CEPH OSD)

RAM: 4 x 256GB
CPU: 2x 10 core Xeon E5-2650
v3 2.2GHz
Hyperconverged Storage:
1 x 400GB SSD SAS (local)
6 x 1TB 7.2K SAS (CEPH OSD)

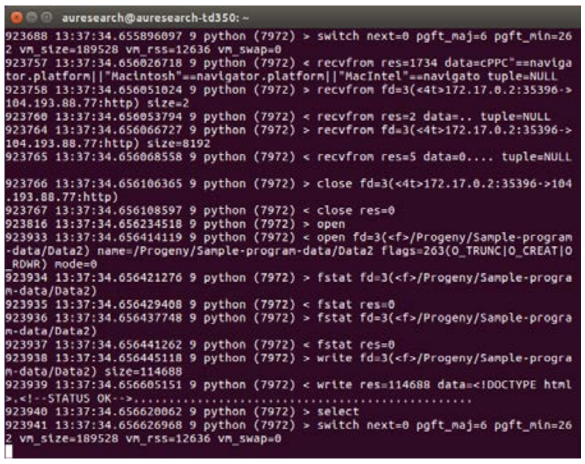
Illustration 11 – Cloud Hardware

## 2. Post Deployment

Once our Openstack cloud was deployed, we setup the virtual networks necessary to grant virtual machines operating on our cloud access to the Internet. This was necessary because of our lack of allotted floating IPs. We then stored our Atomic Host image using Openstack Glance so that it could be provisioned. We then started our virtual machine using the Openstack Horizon dashboard and installed all necessary software such as Sysdig onto the virtual machine. To demonstrate that our solution worked it was necessary to exemplify running a contained application and performing live introspection capabilities. Once the necessary software was installed, we pulled some sample Docker images from our DockerHub registry and began deploying a contained sample application. Once the application was running on our virtual infrastructure, we were able to introspect using Sysdig and see system calls along with other information.


auresearch@auresearch-td350: ~/Documents
File Edit View Search Terminal Help

```
Viewing: Processes For: whole machine
Source: Live System Filter: evt.type!=switch
Select View          Processes
Connections          This is the typical top/htop process list, showing usage of
Containers           resources like CPU, memory, disk and network on a by process
Containers Errors     basis.
Directories
Errors               Tips
File Opens List      This is a perfect view to start a drill down session. Click
Files                enter or double click on a process to dive into it and explo
I/O by Type          re its behavior.
K8s Controllers
K8s Namespaces       Columns
K8s Pods             PID: Process PID.
K8s Services         CPU: Amount of CPU used by the proccess.
New Connections      USER:
Page Faults          TH: Number of threads that the process contains.
Processes            VIRT: Total virtual memory for the process.
Processes CPU        RES: Resident non-swapped memory for the process.
Processes Errors     FILE: Total (input+output) file I/O bandwidth generated by t
Processes FD Usage   he process, in bytes per second.
Server Ports         NET: Total (input+output) network I/O bandwidth generated by
Socket Queues         the process, in bytes per second.
Spectrogram-File     Command: The full command line of the process.
Spy Syslog
Spy Users            ID
System Calls         procs
Threads
F1Help  F2Views F4FilterF5Echo  F6Dig   F7LegendF8ActionsF9Sort  F12SpectroCTRL+
```



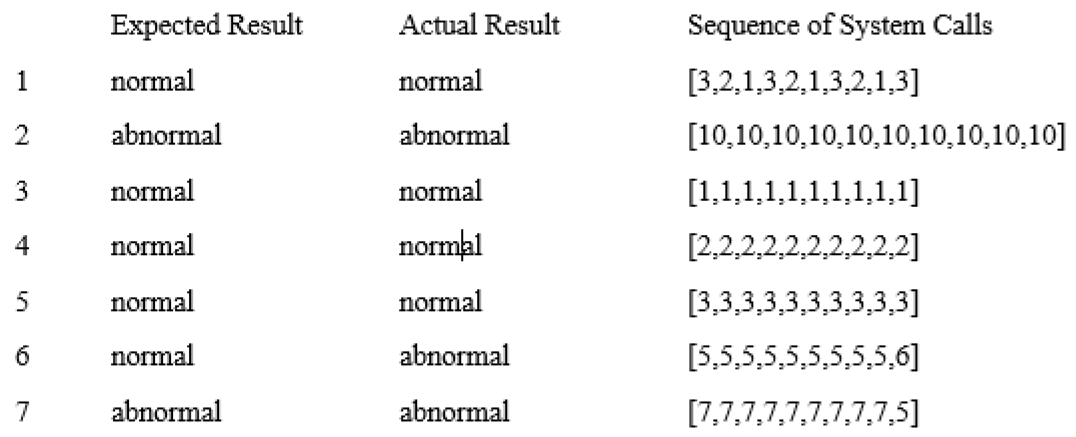
```
923688 13:37:34.655896097 9 python (7972) > switch next=0 pgft_maj=6 pgft_min=26
2 vm_size=189528 vm_rss=12636 vm_swap=0
923757 13:37:34.656026718 9 python (7972) < recvfrom res=1734 data=cPPC"==naviga
tor.platform||"Macintosh"==navigator.platform||"MacIntel"==navigato tuple=NULL
923758 13:37:34.656051024 9 python (7972) > recvfrom fd=3(<4t>172.17.0.2:35396->
104.193.88.77:http) size=2
923760 13:37:34.656053794 9 python (7972) < recvfrom res=2 data=.. tuple=NULL
923764 13:37:34.656066727 9 python (7972) > recvfrom fd=3(<4t>172.17.0.2:35396->
104.193.88.77:http) size=8192
923765 13:37:34.656068558 9 python (7972) < recvfrom res=5 data=0.... tuple=NULL

923766 13:37:34.656106365 9 python (7972) > close fd=3(<4t>172.17.0.2:35396->104
.193.88.77:http)
923767 13:37:34.656108597 9 python (7972) < close res=0
923816 13:37:34.656234518 9 python (7972) > open
923933 13:37:34.656414119 9 python (7972) < open fd=3(<f>/Progeny/Sample-program
-data/Data2) name=/Progeny/Sample-program-data/Data2 flags=263(O_TRUNC|O_CREAT|O
_RDWR) mode=0
923934 13:37:34.656421276 9 python (7972) > fstat fd=3(<f>/Progeny/Sample-progra
m-data/Data2)
923935 13:37:34.656429408 9 python (7972) < fstat res=0
923936 13:37:34.656437748 9 python (7972) > fstat fd=3(<f>/Progeny/Sample-progra
m-data/Data2)
923937 13:37:34.656441262 9 python (7972) < fstat res=0
923938 13:37:34.656445118 9 python (7972) > write fd=3(<f>/Progeny/Sample-progra
m-data/Data2) size=114688
923939 13:37:34.656605151 9 python (7972) < write res=114688 data=<!DOCTYPE html
>.<!--STATUS OK-->.........................................
923940 13:37:34.656620062 9 python (7972) > select
923941 13:37:34.656626968 9 python (7972) > switch next=0 pgft_maj=6 pgft_min=26
2 vm_size=189528 vm_rss=12636 vm_swap=0
```

Illustration 12 – Sysdig GUI (left) and Monitoring Information (right)

## 3. Intelligent Agent

Although work on the intelligent agent is continuing, it has already been able to demonstrate some results. Currently, our intelligent agent has begun to classify sequences that represent system calls and determine if they are normal or abnormal when trained on labeled sequences. Below we can see some sample predictions from our intelligent agent. In this example, we limit the number of system calls an application can perform to ten. From there we divide the system calls into three different groups: system calls 1-5 are allowed in any order, 6 is allowed but not in any order, and the rest of the system calls are forbidden. Using these rules, we trained and ran predictions on the following system calls. From this we can see that it successfully grouped most sequences. The only sequence that did not classify correctly was the sixth sequence. Five is always allowed but six is not necessarily allowed in any order. This is most likely due to lack of training data concerning allowed system calls and provides an area for improvement.



| | Expected Result | Actual Result | Sequence of System Calls |
|---|---|---|---|
| 1 | normal | normal | [3,2,1,3,2,1,3,2,1,3] |
| 2 | abnormal | abnormal | [10,10,10,10,10,10,10,10,10,10] |
| 3 | normal | normal | [1,1,1,1,1,1,1,1,1,1] |
| 4 | normal | normal | [2,2,2,2,2,2,2,2,2,2] |
| 5 | normal | normal | [3,3,3,3,3,3,3,3,3,3] |
| 6 | normal | abnormal | [5,5,5,5,5,5,5,5,5,6] |
| 7 | abnormal | abnormal | [7,7,7,7,7,7,7,7,7,5] |

Illustration 13 – Results from Sample Run of Intelligent Agent

# CHAPTER 5 – Conclusion and Future Work

The solution presented meets the requirements set forth by our customer. The solution:

- Presents a comprehensive virtualization plan that is centered around security and promoting isolation between different layers of virtualization.
- Provides a general monitoring solution that could work on any Linux system and that provides diverse and detailed monitoring data.
- Presents a private cloud that serves as a management solution for the proposed virtualization plan that can provision virtual machines when needed.
- Provides a DevSecOps process that governs cloud and deployment operations.

- Presents the plan for an intelligent agent that is currently in development and has shown significant progress.

With these requirements met, our customer is provided a model for success towards creating their own virtual infrastructure. Concepts such as private cloud deployment were not considered by our customer before this research began and was a highly promising solution for their situation. Our customer was also not aware of the benefits of container virtualization and how well they contained. Because of this research, we eliminated confusion that could result in containers being passed over and instead proved their ability to improve military systems. Now that these deliverables have been presented, our customer is still interested in pursuing the development of an intelligent agent to detect malicious behavior. This is the first step into machine learning for our customer and allows them to start incorporating artificial intelligence to make more sophisticated systems.

## 1. Future Work

The current way forward is to continue with the development of the intelligent agent. Currently, we have a three-point plan to make progress:

- Improve training data to better differentiate sequences of system calls.
- Explore and deepen the network to be able to recognize complex features.
- Increase realism as progress is made.

Formulating and generating training data is one of the most important steps because it is the main way of teaching our intelligent agent to discern good application behaviors from bad. There are hundreds of system calls in the Linux kernel and training our agent on every permutation seems infeasible. Therefore, current research is working towards how to best train the agent.

Once we have a better grasp on generating higher quality training data, we can then experiment and deepen the network to achieve better results and determine more features from our data. Along with supervised learning, unsupervised learning and clustering also present a promising option to group normal behaviors from abnormal and give more insight to the actions of running applications.

As we achieve greater results when classifying sequence of system calls, it will be necessary to branch out and incorporate other monitoring information. CPU utilization, memory utilization, and network IO all present opportunities for our intelligent agent to learn and provide more sophisticated monitoring capabilities.

## Bibliography

[Xavier, 2013] M. G. Xavier, M. V. Neves, F. D. Rossi, T. C. Ferreto, T. Lange and C. A. F. De Rose, "Performance Evaluation of Container-Based Virtualization for High Performance Computing Environments," *2013 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, Belfast, 2013, pp. 233-240.
doi: 10.1109/PDP.2013.41
URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6498558&isnumber=6498515

[Man Namespaces] Anon. User Namespaces. Retrieved May 2, 2017 from http://manpages.ubuntu.com/manpages/wily/man7/user_namespaces.7.html

[Man Cgroups] Anon. Cgroups. Retrieved May 2, 2017 from http://manpages.ubuntu.com/manpages/yakkety/man7/cgroups.7.html

[Man Seccomp] “Seccomp(2) - Linux Manual Page.” *Access(2) - Linux Manual Page*, man7.org/linux/man-pages/man2/seccomp.2.html.

[Man Capabilities] “Capabilities(7) - Linux Manual Page.” *Access(2) - Linux Manual Page*,

[Docker, 2017] Anon. Docker Overview. Retrieved May 2, 2017 from https://docs.docker.com/engine/docker-overview/

[Docker, 2017b] “About Storage Drivers.” *Docker Documentation*, 28 June 2018, docs.docker.com/storage/storagedriver/#container-and-layers.

[Docker, 2017c] “Company.” *Docker*, 28 Dec. 2017, www.docker.com/company.

[Hines, 2015] Hines, Chris. “Docker Basics Webinar Q&A: Understanding Union Filesystems, Storage and Volumes.” *Docker Blog*, 8 Oct. 2015, blog.docker.com/2015/10/docker-basics-webinar-qa/.

[Pahl, 2015]Pahl, Claus. "Containerization and the paas cloud." *IEEE Cloud Computing* 2.3 (2015): 24-31.

[Garfinkel, 2003] Tal Garfinkel and Mendel Rosenblum. 2003. A Virtual Machine Introspection Based Architecture for Intrusion Detection. In *NDSS*. Retrieved from http://dblp.uni-trier.de/db/conf/ndss/ndss2003.html#GarfinkelR03

[Mell, 2011] Mell, Peter, and Tim Grance. "The NIST definition of cloud computing." (2011).

[NVIDIA, 2018] “Deep Learning.” *NVIDIA Developer*, 6 July 2018, developer.nvidia.com/deep-learning.

[Felter, 2015] Felter, Wes, et al. "An updated performance comparison of virtual machines and linux containers." *Performance Analysis of Systems and Software (ISPASS), 2015 IEEE International Symposium On*. IEEE, 2015.

[Joy, 2015]Joy, Ann Mary. "Performance comparison between linux containers and virtual machines." *Computer Engineering and Applications (ICACEA), 2015 International Conference on Advances in*. IEEE, 2015.

[RedHat] Red Hat, Inc. “Introduction to Project Atomic.” *Project Atomic*, www.projectatomic.io/docs/introduction/.

[RedHat, 2017] Differences between RHEL Server and RHEL Atomic Host.” *A Brief History of Cryptography - Red Hat Customer Portal*, 31 May 2017, access.redhat.com/articles/2772861.

[Walsh, 2014] Walsh, Dan. “Yet Another Reason Containers Don't Contain: Kernel Keyrings.” *Project Atomic*, 15 Sept. 2014, www.projectatomic.io/blog/2014/09/yet-another-reason-containers-don-t-contain-kernel-keyrings/.

[Price, 2004] Price, Daniel, and Andrew Tucker. "Solaris Zones: Operating System Support for Consolidating Commercial Workloads." *LISA*. Vol. 4. 2004

[Chandramouli, 2017] Chandramouli, Ramaswamy. *Security Assurance Requirements for Linux Application Container Deployments*. US Department of Commerce, National Institute of Standards and Technology, 2017.

[Lopez, 2015] Lopez, Antonio. “Linux Capabilities and How to Avoid Being Root.” *CERTSI*, 4 June 2015, www.certsi.es/en/blog/linux-capabilities-en.

[Souppaya, 2017] Souppaya, Murugiah, John Morello, and Karen Scarfone. "Application Container Security Guide." *NIST Special Publication* 800 (2017): 190.

[Grance, 2010] Mell, Peter, and Tim Grance. "The NIST definition of cloud computing." *Communications of the ACM* 53.6 (2010): 50.

[Hashizume, 2013] Hashizume, Keiko, et al. "An analysis of security issues for cloud computing." *Journal of internet services and applications* 4.1 (2013): 5.

[Grobauer, 2011] Grobauer, Bernd, Tobias Walloschek, and Elmar Stocker. "Understanding cloud computing vulnerabilities." *IEEE Security & Privacy* 9.2 (2011): 50-57.

[Ebert, 2016] Ebert, Christof, et al. "DevOps." *IEEE Software* 33.3 (2016): 94-100.

[Fitzgerald, 2017] Fitzgerald, Brian, and Klaas-Jan Stol. "Continuous software engineering: A roadmap and agenda." *Journal of Systems and Software* 123 (2017): 176-189.

[Cois, 2014] Cois, C. Aaron. “A Generalized Model for Automated DevOps.” *Carnegie Mellon University SEI Insights*, SEI Blog, 2 June 2014, insights.sei.cmu.edu/sei_blog/2014/06/a-generalized-model-for-automated-devops.html.

[Rahman, 2016] Rahman, Akond Ashfaque Ur, and Laurie Williams. "Software security in devops: synthesizing practitioners’ perceptions and practices." *Continuous Software Evolution and Delivery (CSED), IEEE/ACM International Workshop on*. IEEE, 2016.

[Stackalytics] “Official Community Contribution during OpenStack Rocky Release.” *Stackalytics*, stackalytics.com/.

[Kolla] “Welcome to Kolla-Ansible's Documentation!.” *OpenStack Docs: Introduction to OpenStack*, 22 June 2018, docs.openstack.org/kolla-ansible/latest/.

[Ansible] “Installation Guide.” *Desired State Configuration - Ansible Documentation*, 14 June 2018, docs.ansible.com/ansible/2.5/installation_guide/intro_installation.html#basics-what-will-be-installed.

[Carroll, 2011] Carroll, Mariana, Alta Van Der Merwe, and Paula Kotze. "Secure cloud computing: Benefits, risks and controls." *Information Security South Africa (ISSA), 2011*. IEEE, 2011.

[Younge, 2010] Younge, Andrew J., et al. "Efficient resource management for cloud computing environments." *Green Computing Conference, 2010 International*. IEEE, 2010.

## Appendix 1 – Source Code

### 1. Main Method of Intelligent Agent

```
'''
Created on Jun 27, 2018

@author: rober
'''

from tensorflow.python.keras.models import Sequential
from tensorflow.python.keras.layers import LSTM, Dense
from numpy import array
import numpy
from app.TrainingDataCreator import TrainingDataCreator
from app.BehaviorCreator import BehaviorCreator


def main():
    sampleSize = 10;#313;
    numberOfSamples = 1;
    numberOfFeatures = 1;
    numberOf64BitSystemCalls = 20;#313;
    numberOfClasses =    2;



    trainingDataCreator = TrainingDataCreator();
    allowedSystemCalls = trainingDataCreator.createArrayFromRange(1, 6); #1-20
    allowedSystemCallsInAnyOrder = trainingDataCreator.createArrayFromRange(1, 5);
#1-10
    allowedSequences = [];
    badSequences =
trainingDataCreator.getLongRepeatedSequenceOfSystemCalls(trainingDataCreator.createAn
ArrayFromOneToNumberOfSystemCalls());
    train_x, train_y, test_x, test_y =
trainingDataCreator.createData(allowedSystemCalls, allowedSystemCallsInAnyOrder,
allowedSequences, badSequences);

    #train_x = numpy.random.random((1000, 3, 1));
    #train_y = numpy.random.random((1000, numberOfClasses));
    print(train_x);
    print(train_x.shape)
    model = buildModel(numberOfSamples, sampleSize, numberOfFeatures);
    model.compile(loss="binary_crossentropy", optimizer="adam",
metrics=['accuracy']);
    model.fit(train_x, train_y, batch_size=16, epochs=10);
```

```
 score = model.evaluate(test_x, test_y, batch_size=16);
    print(score);

    print(model.predict_classes(numpy.array([3,2,1,3,2,1,3,2,1,3]).reshape(1,10,1)));

print(model.predict_classes(numpy.array([10,10,10,10,10,10,10,10,10,10]).reshape(1,10
,1)));
    print(model.predict_classes(numpy.array([1,1,1,1,1,1,1,1,1,1]).reshape(1,10,1)));
    print(model.predict_classes(numpy.array([2,2,2,2,2,2,2,2,2,2]).reshape(1,10,1)));
    print(model.predict_classes(numpy.array([3,3,3,3,3,3,3,3,3,3]).reshape(1,10,1)));
    print(model.predict_classes(numpy.array([5,5,5,5,5,5,5,5,5,6]).reshape(1,10,1)));
    print(model.predict_classes(numpy.array([7,7,7,7,7,7,7,7,7,5]).reshape(1,10,1)));


def createSample(sampleSize):
    array1 = [1, 2, 3];
    sequence = [];

    for i in range(sampleSize):
        sequence.extend(array1);

    data = array(sequence);

    numberOfSamples = 1;
    numberOfFeatures = 1;
    data = data.reshape(numberOfSamples, sampleSize * 3, numberOfFeatures)
    return data;

def buildModel(numberOfSamples, sampleSize, numberOfFeatures):
    model = Sequential();
    model.add(LSTM(32, input_shape=(sampleSize, numberOfFeatures)));
    model.add(Dense(2, activation='sigmoid'));
    return model;

def generateGoodBehaviors(self):
    goodBehaviorSequences = [];

    behaviorCreator = BehaviorCreator();
    behaviorCreator.addBehavior([51, 52, 53]);
    goodBehaviorSequences.append(behaviorCreator.getBehavior());
    behaviorCreator.addBehavior([61, 62, 63]);
    goodBehaviorSequences.append(behaviorCreator.getBehavior());
    behaviorCreator.addBehavior([81, 82, 83]);
    goodBehaviorSequences.append(behaviorCreator.getBehavior());



if __name__ == '__main__':
  main();
```

## 2. Training Data Creator

```
'''
Created on Jun 30, 2018

@author: rober
'''
import itertools
from random import shuffle
import numpy as np

class TrainingDataCreator(object):

    numberOfSystemCalls = 10;#313

    def __init__(self):
        pass

    def createData(self, listOfAllowedSystemCalls, 
listOfAllowedSystemCallsThatCanBeInAnyOrder, listOfAllowedSequences, 
listOfBadSequences):

        notAllowedSystemCalls = 
self.getArrayOfNotAllowedSystemCalls(listOfAllowedSystemCalls);
        someNotAllowedSystemCalls = self.createArrayFromRange(5, 10)

        badPermutations = [];
        if not len(listOfBadSequences) == 0:
            badPermutations.extend(listOfBadSequences);

        goodPermutations = [];
        if not len(listOfAllowedSequences) == 0:
            goodPermutations.extend(listOfAllowedSequences);

        #allPossiblePermutationsWithoutRepetition = 
itertools.permutations(self.createAnArrayFromOneToNumberOfSystemCalls());
        allPossiblePermutationsOfNotAllowedSystemCalls = 
itertools.permutations(someNotAllowedSystemCalls);
        allPossiblePermutationsOfNotAllowedSystemCalls = 
self.extendPermutationsToFitSequenceSize(allPossiblePermutationsOfNotAllowedSystemCal
ls, self.numberOfSystemCalls);
        #badPermutations.extend(allPossiblePermutationsWithoutRepetition);
        badPermutations.extend(allPossiblePermutationsOfNotAllowedSystemCalls);

badPermutations.extend(self.getSequencesOfPurelyDisallowedSystemCalls(notAllowedSyste
mCalls));

        allPossiblePermutationsOfSystemCallsThatCanBeInAnyOrder = 
itertools.permutations(listOfAllowedSystemCallsThatCanBeInAnyOrder);
        allPossiblePermutationsOfSystemCallsThatCanBeInAnyOrder = 
self.extendPermutationsToFitSequenceSize(allPossiblePermutationsOfSystemCallsThatCanB
eInAnyOrder, self.numberOfSystemCalls);

goodPermutations.extend(allPossiblePermutationsOfSystemCallsThatCanBeInAnyOrder);
```

```
        goodLabels = self.createLabels(True, goodPermutations);
        badLabels = self.createLabels(False, badPermutations)

        goodPermutations.extend(badPermutations);
        goodLabels.extend(badLabels);

        #shuffeledTrainingData, shuffeledLabels =
self.unison_shuffled_copies(goodPermutations, goodLabels);

        percentageOfTrainingData = .8
        x_train, x_test = self.splitArrayByPercentage(goodPermutations,
percentageOfTrainingData);
        y_train, y_test = self.splitArrayByPercentage(goodLabels,
percentageOfTrainingData)

        self.temporarilyAppendDifferentSequencesToTestData(x_test, y_test);


        x_train = np.array(x_train);
        y_train = np.array(y_train);
        x_test = np.array(x_test);
        y_test = np.array(y_test);

        x_train = x_train.reshape(x_train.shape[0], x_train.shape[1], 1);
        x_test = x_test.reshape(x_test.shape[0], x_test.shape[1], 1);

        print(x_train.shape);
        print(x_test.shape);
        print(y_train.shape);
        print(y_test.shape);

        return x_train, y_train, x_test, y_test

    def temporarilyAppendDifferentSequencesToTestData(self,x_test, y_test):
        goodLabel = [0, 1];
        badLabel = [1,0];

        x_test.append([2,1,2,1,2,1,2,1,2,1]);
        x_test.append([1,2,3,1,2,3,1,2,3,1]);
        x_test.append([1,2,3,4,1,2,3,4,1,2]);
        x_test.append([1,2,3,4,5,1,2,3,4,5]);
        for i in range(4):
            y_test.append(goodLabel);


        x_test.append([1,2,3,4,5,6,1,2,3,4]);
        x_test.append([1,2,3,4,5,6,7,1,2,3]);
        x_test.append([1,1,1,1,1,1,1,1,1,10]);
        x_test.append([1,1,1,1,1,10,10,10,10,10]);
        for i in range(4):
            y_test.append(badLabel);

    def extendPermutationsToFitSequenceSize(self, permutations, sequenceSize):
        listOfFittedPermutations = []
        for permutation in permutations:
            permutationInListForm = list(permutation);
            permutationInListForm =
self.repeatSequenceToLimitAndPad(permutationInListForm, sequenceSize, 0);
            listOfFittedPermutations.append(permutationInListForm);

        return listOfFittedPermutations;


    def createLabels(self, isGood, data):
        labels = [];

        if isGood:
            goodLabel = [0, 1];
            for i in range(len(data)):
                labels.append(goodLabel);
        elif not isGood:
            badLabel = [1,0];
            for i in range(len(data)):
                labels.append(badLabel);
        else:
            raise Exception()

        return labels;

    def splitArrayByPercentage(self, array, percentage):

        if percentage < 0 or percentage > 1:
            raise Exception();


        firstSplitMark = int(len(array) * percentage);

        firstPortion = [];

        for i in range(firstSplitMark):
            firstPortion.append(array[i]);

        secondPortion = []
        for j in range(firstSplitMark, len(array)):
            secondPortion.append(array[j]);

        return firstPortion, secondPortion


    def createAnArrayFromOneToNumberOfSystemCalls(self):
        return self.createArrayFromRange(1, self.numberOfSystemCalls)

    def createArrayFromRange(self, begin, end):
        if begin == end:
            array = []
            array.append(begin);
            return array;
        if end < begin:
            raise Exception();

        array = [];
        for i in range(begin, end + 1):
            array.append(i);

        return array


    def getArrayOfNotAllowedSystemCalls(self, listOfAllowedSystemCalls):
        notAllowedSystemCalls = self.createAnArrayFromOneToNumberOfSystemCalls();

        for allowedSystemCall in listOfAllowedSystemCalls:
            notAllowedSystemCalls.remove(allowedSystemCall)

        return notAllowedSystemCalls


    def getSequencesOfPurelyDisallowedSystemCalls(self, notAllowedSystemCalls):
        sequencesOfPurelyDisallowedSystemCalls = [];
        for notAllowedSystemCall in notAllowedSystemCalls:
            tempSequence = []
            for i in range(self.numberOfSystemCalls):
                tempSequence.append(notAllowedSystemCall);
            sequencesOfPurelyDisallowedSystemCalls.append(tempSequence);

        return sequencesOfPurelyDisallowedSystemCalls;


    def getLongRepeatedSequenceOfSystemCalls(self, systemCalls):
        longRepeatedSequencesOfSystemCalls = [];

        for systemCall in systemCalls:
            oneSequence = [];
            for i in range(self.numberOfSystemCalls):
                oneSequence.append(systemCall);

            longRepeatedSequencesOfSystemCalls.append(oneSequence);

        return longRepeatedSequencesOfSystemCalls;

    def repeatSequenceToLimitAndPad(self, sequence, limit, paddingNumber):
        sequenceToRepeat = sequence.copy();
        numberOfRepetitions = int(limit / len(sequence));

        newSequence = [];
        for i in range(numberOfRepetitions):
            newSequence.extend(sequenceToRepeat);

        paddingAmount = limit % len(sequence);

        for i in range(paddingAmount):
            newSequence.append(paddingNumber);
        return newSequence;


    def simulateNormalPerformance(self):
        pass
```

## 3. Docker Installer

```
sudo apt-get remove docker docker-engine


sudo apt-get update


sudo apt-get install \
    apt-transport-https \
    ca-certificates \
    curl \
    software-properties-common


curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -


sudo apt-key fingerprint 0EBFCD88
echo "Script runner, verify that the key fingerprint is: 9DC8 5822 9FC7 DD38 854A E2D8 8D81 803C 0EBF
CD88"


sudo add-apt-repository \
   "deb [arch=amd64] https://download.docker.com/linux/ubuntu \
   $(lsb_release -cs) \
   stable"


sudo apt-get update


sudo apt-get install docker-ce
```

## 4. Dockerfile for Sample C++ Application

```
FROM ubuntu:latest



RUN apt-get update
RUN apt-get -y install git

RUN git clone https://github.com/rca0007/DockerSamples.git

RUN apt-get -y install g++
RUN apt-get -y install make
RUN make -C /DockerSamples/SampleCPPForDocker/OpenBash

CMD ./DockerSamples/SampleCPPForDocker/OpenBash
```

## 5. C++ Application That Opens Bash Shell

```
#include<unistd.h>
#include<stdlib.h>

using namespace std;

int main(int len, char** args)
{
        static char* bash[2];
        bash[0] = (char *) "/bin/bash";
        bash[1] = NULL;
        execv(*bash, bash);
}
```