ManaGRR

Scaling and Managing GRR Rapid Response

GRR as a Service

Abstract

GRR Rapid Response is an application for incident response and live forensics. It involves dealing with separate servers communicating together in a secure manner. Using this application for production use is quite difficult to manage when dealing with multiple clients. The result is a python-flask application that communicates with Proxmox hypervisor to provision each virtual role with a proper private network stack. It expands GRR's capability and ease-of-setup especially when dealing with a growing business.

Kevin Law
@thatarchguy
mail@kevinlaw.info

Contents

Introduction

Digital Forensic tools have evolved since the field was established to a more autonomous and distributed platform. Instead of manually imaging and searching for evidence, software can perform those repetitive steps. Some have been written to scale and provide service to multiple sources of evidence across a network. These systems are expensive and proprietary solutions. Installing this software from paid vendors is generally very straightforward and does not require any real knowledge of systems administration. Support is readily available because it's included in the price. However, they do not necessarily meet the needs of the user, and the user cannot change the software.

GRR Rapid Response (GRR) is a tool for distributed forensics that is free and open source. Anyone can copy the source and modify it to their desires. The community authors the software, and can add in its own modifications. GRR has better features than its competitors, but is difficult to setup and manage.

If a product is difficult to setup and manage, a user will decide it is not worth the trouble and pay a company for a better solution. GRR is just a tool; it was not designed to maintain itself and the underpinnings of its framework. It has the potential to be a tool used by forensic investigators to offer as a service to clients. It just needs to be easily scalable, replicable, and customizable for each client.

Since the project is open source, the only support available is from asking the community of developers for help on an issue. This can be a concern if a company is selling this as their service and a problem does arise. Most likely an issue posted will be solved by other

members of the community and the fix is patched into the code. This is not a model that proprietary software utilizes, and some wait months to push out an update with patches to problems they found.

If the setup and management part of GRR is solved, then the tool will be used in the field much more. It wouldn't cost the company any money for the software, and they can begin selling it as a service with little overhead. Since GRR is such a powerful tool, the digital forensics field will benefit greatly from it.

Background Research

History

Digital Forensics is a rapidly growing field. The forensic tools that are used to analyze modern user devices are getting more advanced. The user devices that are being created are becoming more intricate and complicated. Updating tools and methodologies will give an investigator the upper hand when handling new devices. In order to understand all the features of modern forensic software, we must look at the tools that led to their creation.

Digital Forensics was created out of need when offices shifted from paper to computers. Employees were given computers, and problems started to arise. The IT department would perform a "live analysis" of the machine by just going through the actual computer files manually. Shortly after, it was realized that bit-by-bit images could be taken of the hard drives using developed tools such as dd or SafeBack (Charters). These images could be taken somewhere else for analyzing without tampering with the original data.

A formal process did not exist for Digital Forensics, as it was still the job of the IT department of the company. This led to evidence found and used in court being dismissed for inappropriate use because of the lack of handling and collecting (Charters).

Basic tools for automating pulling common data out of systems were created. RegRipper is a tool that will process a Window's registry hive and print out information like user names, install date, and other useful information. Exif Tool was created to read the metadata on a variety of file types. Volatility can capture the memory of a computer for analysis. Computer

memory is volatile and can hold very important information that will be lost on shutdown. Software like these greatly reduced the time needed to analyze a system because they automating the process.

It was known that bugs will arise in software and the need for reliability testing and ratings were needed. It was common for these tools to become open source for reasons other than cost. Open source tools allow anyone to read the source and understand the procedures. A user can verify that the tool is following the published process and not doing only the minimum required (Carrier). Having other programmers see the source code and review it made a much more robust tool and any algorithms could be properly documented and defended against in court.

Larger digital forensics tool suites such as Encase and FTK are tailored to the enterprise side of forensics. Imaging, analysis, and reporting could be done in one convenient program. Features like bookmarking, image finders, and report generation are what keep these suites the "go-to" programs. Basic processes such as registry analysis and image file extraction could be automated in these tools. EnCase Enterprise is currently the industry standard for Forensic investigation (Guidance Software). These suites are popular tools for Forensic companies, but they come at a great cost: Encase Enterprise is currently $3,995 per seat (Guidance Software).

Automated tools lessen redundancy of running multiple tools on an image and correlating all the results together. Software like MantaRay takes common tools used in an analysis and brings them all together in a suite that gathers all the results together in an easy to

read timeline and report. Further analysis can be done if significant results are found, but this tool gives an excellent overview into an image with little time and input from the user.

Accessing a machine remotely is the latest trend in Forensics. “Remote live forensics has recently been increasingly used in order to facilitate rapid remote access to enterprise machines” (Cohen). The process can allow an investigator to analyze as soon as an incident happens, rather than wait for the device to be transported and imaged in the lab. F-Response is a tool that can mount a remote filesystem to be analyzed with tools such as Encase. Mandiant Intelligent Response (MIR) can react to an incident and allow artifacts left behind to be gathered. Remote tools are parallel with incident response procedures and can benefit both fields. The downside is that these tools require an active connection.

Live forensics has grown popular in the past few years. Significant data can be found on many different parts of a computer. Some of these parts do not store data long term and can be lost before an investigator gets a chance to find it. Loss of valuable data during a forensic investigation can be damaging to the overall investigation itself, and can make the difference between finding key evidence and letting it get away. The loss of data is a normal occurrence on a computer due to the way information is stored. Important Information such as encryption keys or passwords is only stored in the computer’s RAM for small amount of time and is cleared as soon as that computer is powered off (Cohen).

The model of live forensics ensures the ability to gather information from a computer instantaneously. Artifacts in RAM and files on the hard drive are easily accessible, and forensic

investigators are given the opportunity to collect more information in less amount of time (Cohen).

GRR Rapid Response

GRR Rapid Response "is an incident response framework focused on remote live forensics" (GRR). It can remotely access filesystems, perform analysis and process information, and do these tasks across an entire network of computers automatically. The framework was written by employees at Google in python and can remotely analyze Windows, Macintosh, and Linux operating systems. It is still in heavy development and only beta versions have been released.

"The primary motivation was that we felt the state of the art for incident response was going in the wrong direction, and wasn't going to meet our cross platform, scalability, obfuscation or flexibility goals for an incident response agent" (GRR-Doc). GRR focuses on incident response, the approach to managing the aftermath of an attack, but the techniques used for analyzing systems are forensically sound and evidence gathered will hold in a court of law.

The GRR framework requires multiple server roles to be utilized in order to handle the amount of information coming in. An enroller accepts connections from computers and enrolls them into GRR. A database role handles all of the information from the connected computers. Figure 1 shows the different roles and how they interact. The entire system revolves around this database which is used to store an AFF4 (Advanced Forensic Format 4) evidence file, which contains all the information collected from the GRR Clients. The database and resultant AFF4 file is accessible to investigators through the web console, or through the more robust command line console. Through both of these interfaces information can be downloaded out of

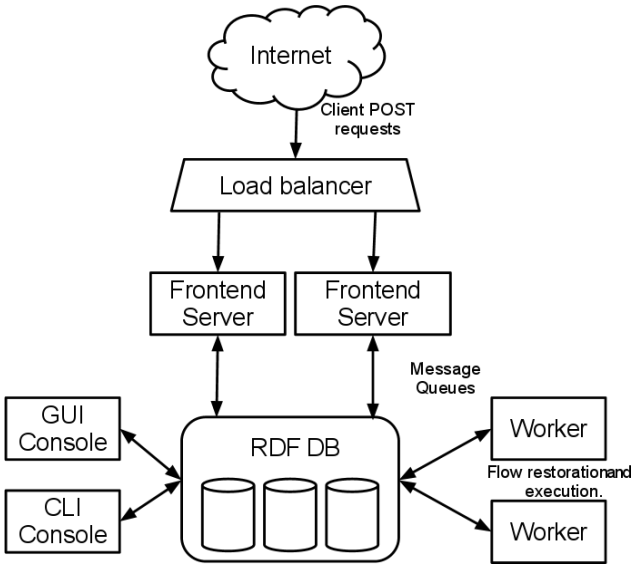


Figure 1 – GRR Framework. Workers are servers reading from the database

the data for further local analysis (Jaffe). The web interface ties everything together in a simple view, and allows one to create flows (jobs for individual computers) and hunts (jobs to be run on a set of computers). Worker roles read work requests from the database and process information. The more computers on the network, the more workers needed.

GRR and similar software solutions do not have a process for per-client individuality in the software. Easy to scale features are not available either. For example, if one wanted to sell GRR as a service, they would have to recreate this GRR framework for every new client. This means a mess of servers, networking, and most importantly management. Software like this are designed for one-use one-client only.

It is not possible for a user to create an automated and scalable solution for similar enterprise software that is offered. GRR is currently the only software of its kind that is open source. This allows for an easy to work with framework to create a management system to offer per-client implementations of GRR. This type of development work is not possible on proprietary software suites.

In order to scale a service, the roles first need to be isolated from the core and networked together. Scaling provides a way the handle growth. GRR requires workers based on the demand of the client. Providing a means for this to be as simple as possible will make GRR

scalable to any client needs. GRR has clear documentation on what each role is and how it receives information from the interface. Each role requires its own operating system and hardware to run on once separated. Separation of the code dependencies for a smaller footprint and optimization would be ideal, but not necessary. The roles should have little bottleneck in communication with each other, and should be able to run normally without major changes to the code base. Figure 1 still applies, but the connections between the roles are now on a different medium.

Once the scalability problem is solved, duplicating roles for redundancy or performance can now be performed. Multiple database roles can be made for failover just in case the primary database fails. This is key for hosting a service in a production environment. The worker roles can be created without duplicating the entire GRR code base, and can be installed as needed.

Replicating the entire process is what will allow per-client individualization. It will also be the key in order to generate multiple clusters of the scaled GRR framework for clients as needed. This has not been done before. Any attempts made involved manually creating the servers and manually networking them together. The installations need to be automated and pluggable down to the hardware and operating system level. The hardware has to be customized to network the roles together, and the connectivity be reflected in the operating system. The GRR install of each role would need a way to be custom installed with its cluster and client specifications. New roles like workers need to have an easy way of adding itself to the cluster and removing itself when needed. Currently a few products were developed that can assist with this process.

Software

Puppet is an IT automation software that can perform operations on an unlimited number of machines. “Once a systems administrator defines the software and configuration of a particular machine, Puppet Enterprise can automate that template across dozens or even thousands of other machines” (Griffin). It requires a server to run, and requires learning a language and structure for the template files. Chef is a similar piece of software to Puppet. It uses recipes to manage actions, which use a language and structure of its own. Software like this was created for the purpose of managing and configuring thousands of machines for a service. “The need for such tools originated with Google, Amazon.com, and their peers, who have long had to deal with the burden of managing tens or even hundreds of thousands of servers to support vast Web operations” (Kharif, Olga, Ashlee). These tools are very powerful and could be utilized for installing the different roles of GRR automatically.

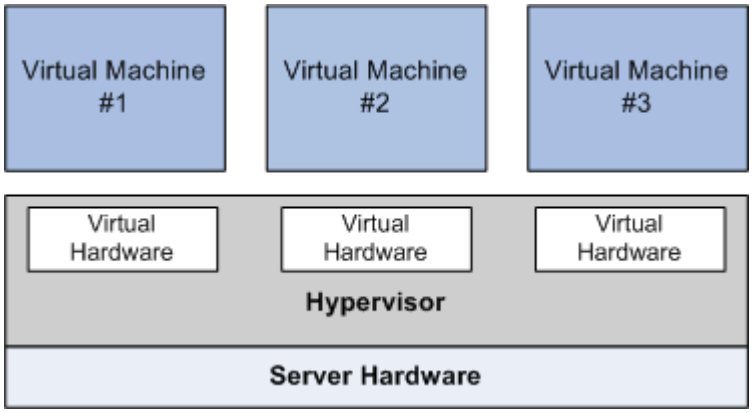


Figure 2 – Virtualization simplified

Virtualization will be vital to replicating and scaling GRR in a cost effective way. Without virtualization, each role would require its own dedicated server. Figure 2 describes how virtualization works to save space by having multiple operating systems running on the same server. Each instance gets its own virtual hardware and is isolated from the other machines. Now a dedicated server can have multiple GRR clusters. This cuts resources

and price needed to run a cluster down significantly. The hardware that the operating system sees can be controlled, monitored, and migrated without any interruption to the service.

Vmware Vsphere is a popular platform for virtualization on dedicated servers. It is commonly used in the industry and is trusted and robust. “Find a major data center anywhere in the world that doesn't use VMware, and then pat yourself on the back because you've found one of the few. VMware dominates the server virtualization market” (Hess). Unfortunately this is an expensive solution and would require licenses for each server. Running virtualization libraries like libvirt with KVM and QEMU would be free, but would be difficult to scale and replicate easily. This route would require more resources and overhead to the project.

Proxmox is a simple and elegant virtualization platform that utilizes modern libraries like KVM, QEMU, and OpenVZ to enable virtualizing many different platforms. Proxmox servers can be clustered together to form datacenters, and machines can be migrated between servers without interruption. It comes with a sleek web interface that allows for control over the entire datacenter. Proxmox is written on top of Linux, and is free and open source. “Proxmox Virtual Environment's source code is published under the free software license GNU AGPL, v3 and thus is freely available via code repository (git) for download, use and share” (Proxmox). Since Proxmox is run on Linux, it has all of familiarity of a standard Linux-based operating system. Proxmox has command-line interface (CLI) that can be used to perform any action that can be done in the webui and more.

The networking in GRR is robust out of the box. The clients connect back to GRR from their network. This eliminates the need for port forwarding on the firewall to communicate.

The packets that are sent are fully encrypted, “GRR communication happens using signed and encrypted protobuf messages. We use 1024 bit RSA keys to protect symmetric AES256 encryption” (GRR-Doc). The communications are very secure and even if captured, they cannot be decrypted. Services like VPN’s will not be required to tunnel the traffic between clients and servers.

The virtual networking interfaces can listen directly for the connection from the clients. The data will be secure and encrypted to prevent anyone from viewing the sensitive information. The inter-cluster connections can be virtual switches in the virtualization platform. “The http server is designed to be exposed to the Internet, but there is no reason for the other components in the GRR system to be” (GRR-Doc). Proxmox supports virtual switching, so the server roles never have to touch outside physical devices when communicating with each other. This ensures a secure connection for the workers to process the data from the database.

The final step for creating an automated management solution for GRR is to wrap all this together with an easy to use management interface to add and manage clients. Web server and database software solutions will be backing the interface. A standard Linux, Apache, Mysql, and PHP (LAMP) stack is a standard way of going about the issue. This is not the case anymore now that better software is being developed.

GRR itself uses Python and MongoDB to run its core. Python started as a scripting language and evolved into being able to run full websites with the help of user-contributed libraries. PHP is a great easy to learn web scripting language. It has simple Mysql interaction with modules, and is included in the LAMP stack. It does not have proper operating system

interaction and scripting capabilities. PHP is trailing behind the web application development world and is slowly being replaced by other languages like Ruby on Rails, Python, and Go. “PHP is Stagnating...Eventually, its one strength, the extremely low barrier to entry, will be nullified as other languages figure out how to be more accessible to the masses” (Montgomery).

Python is flexible enough to script small programs, and powerful enough to run full-fledged web applications even without a standalone web server like Apache. With proper Model-View-Controller (MVC) framework modules implemented, it can become more capable than Apache with PHP. Python-Flask is a framework that enables a developer to create a large web application. It can be paired with SQLAlchemy for object-relational mapping (ORM) a database into python objects. Having the management interface written in Python-Flask means the scripting codebase can be integrated easily and without any system exec commands.

For the management website interface to be easy to deploy and upgrade, Docker will be utilized. Docker is an application that allows for software to be run in a container. This segregates the application from the operating system by creating a virtual machine for it to run in. Docker creates a quick and easy virtual environment utilizing Linux containers. It is very fast to build an application into a container. These containers can be sent to other members of the team for testing and QA. Docker is being used in production to deploy applications more and more now. With operating systems like CoreOS, creating a fleet of deployments is simple.

CoreOS is a Linux-based operating system designed specifically to run Docker containers, and “...built to make large, scalable deployments on varied infrastructure simple to manage“(Ellingwood). It helps abstract the application from the physical servers it runs on. If a

server goes down, CoreOS can automatically move the container around without interruption. It can do this for different containers in the application as well. CoreOS and Docker make the perfect combination for a high-availability production application.

Proxmox can be the host for the virtualized GRR roles for each cluster. Puppet, Chef, or bash scripts can work to automate provisioning and configuring the virtual machines. Python-Flask and SQLAlchemy can be the webserver software and language for the web management interface. The web interface can be thrown into a Docker container and hosted in a CoreOS cluster for high availability.

Currently, this is the only initiative of its kind. Similar programs to GRR are proprietary and do not embrace the need for improvement from the user. Digital Forensics software has evolved from simple tools, to automated analysis, and to what is currently distributed forensics across a network. GRR enables the ability to have forensics in the same step as incident response. With the research gained, a proper management program can be created to utilize GRR more effectively in the field.

Cultural Context

GRR Rapid Response (GRR) is a free and open source framework that allows for the deployment of agents for the purposes of remote distributed forensics and incident response on a network. It can remotely access raw disk and memory of every computer the agent is installed on. Currently the project is in alpha and needs a lot of work. My goal is to create a system that packages GRR Rapid Response into a sellable product. Companies will be able to use my finished product to market distributed forensics and remote incident response to clients. The framework I am creating will be a management system and virtualization wrapper to create per-client virtual GRR clusters with the click of a button. This software will impact the field of digital forensics and change the way we think about incident response.

GRR comprises of many parts that can be separated into different servers for scalability. The database component is most important, as all the others link into it. The workers handle the processing requests from the flow manager. With the idea that workers can be created as needed, my framework allows for quick virtual worker generation in a hosted server cluster or using cloud services.

This will change the way computer forensic companies perform analysis on a network of computers. Setting up a new client takes minutes and just needs them to push the executables to its network. The forensic investigator can start to integrate this tool into their workflow to do a majority of the scripted bulk forensic analysis that would otherwise take days to image and perform on each computer. Since its open source, it's free for anyone to take my system and modify it.

Because of the nature of the data that GRR can produce, it must be handled very securely. Company policies and ethics training will have to be changed for those who have

access to their GRR control panel. Generally only the forensic investigators have access to the panel.

The system will ultimately be used to investigate crimes on company computers, whether it be an internal discrepancy or an outside attacker. It can produce a variety of incriminating information from a computer. Producing this information is not violating anyone's rights in the workplace because there is a lower expectation of privacy. Terms of service agreements protect the use of the software. Most companies already implement monitoring programs that are accordance to their company policy.

Anyone with malicious intent could take my software and modify it for botnet purposes. Modern malware uses load balancers and workers to handle the mass amount of connections and data being processed from its zombie computers. My framework generates these components, and would only take a few hours to integrate a botnet management server into it. This is a risk that many software developers take into consideration. I cannot prevent this, and I cannot control the intent of the users. I will have a user agreement included with the license information of my software.

Since the system will be virtualized onto physical servers, its carbon footprint is much lower than each company running their own GRR instance on physical servers. What would consist of over 5 physical servers per client is reduced to over 5 clients per server on my framework. The power savings will have a drastic impact, because servers require a lot of power to run.

Cultural impacts are an often overlooked factor when developing projects. A project with a positive outlook may end up harming the culture it was meant for. I've looked at the different

impacts my project could have on its community and affecting users, and I have only found a few negative impacts. Legal and social ramifications due to the produced content and abilities of the project may be a cause for concern. The positives greatly outweigh the negatives that cannot be prevented in this case. The digital forensics community will benefit greatly from the integration of my project.

Project Experience and Results

Working with the GRR project has been a pleasant experience. The GRR Mailing lists are very helpful. A Google employee started a blog with tutorials on how to solve common tasks. One of these tutorials that was critical to this project was separating the different roles into their own servers. The GRR project is now hosted on GitHub, so any issues I ran into were quickly dealt with.

Development Experience and Process

The GRR cluster needed to be customized for each client, and the individual machines needed to become aware that they were being externally managed and customized for a client. This took me a while to figure out how to do. One cannot just duplicate GRR-installed virtual machines. The machines would have to be heavily modified to reconfigure GRR from that point. That seemed like it wouldn't be an easy task.

Automating this process was another feat. How do you automate installing an operating system onto a virtual disk? VMware workstation has the easy installer, but that's not open source or easily automated by any means. I managed to find the software Packer. It's an open source tool that allows for provisioning machines images onto the cloud or virtual disks using a simple json file. I used Packer to create the base Ubuntu install image.

Once I had the installation of Ubuntu in an image file, I could then use this as the base to duplicate and customize. I found a software library called libguestfs. This software allows for modifying Linux virtual disk images. It even has a convenient wrapper to sysprep the images in order to generalize them. Libguestfs could generalize, make all the client customizations, and drop a GRR install script to run on boot.

The networking was another obstacle. The hypervisor Proxmox luckily had an API that could be scripted. Each GRR cluster needs its own virtual network. In Proxmox, the virtual network must be a declared virtual interface in /etc/networking/interfaces before the machine could be provisioned. Scripting this to be able to add and remove interfaces was key to getting the network issue solved.

Originally I had a three bash scripts to automate the process of copying the base image, using libguestfs to customize the new image, and provisioning the virtual machine in proxmox with proper networking and the image as the hard drive. This had to be done three times because there are three main parts to GRR: the Database, Controller, and Workers. These original scripts are located in the managrr/provision/legacy folder of my project. This was the basis for my project. Once I had this down, I only had to write a management front end to put a nice UI in front of it.

For the user interface, I chose to write it in python-flask. This is a great framework for using python to create a web interface. SQLite was chosen as the database because it is a single file that is easy to use. ManaGRR is not a database heavy program.

SQLAlchemy, python-flask's database ORM, made creating the database and filling it with data effortless. It reads the configuration from a python class file and can initialize most database engines. I wrapped this entire process with the command "./manage.py createdb".

I wrote a form that could add a client and kickstart the bash scripts. The nodes would be added to the database, and IP addresses would update after they were installed. I wanted this to be scalable, so I gave the option to choose which hypervisor to install the cluster on.

Another issue that sounded easy but turned out to be difficult, “What if the hypervisor is down? How do you test that a hypervisor is up before provisioning?”. If I tried a connection to the hypervisor and it timed out, that timeout would block the whole application. I ended up just sending pings to the hypervisor as a check when the client form is completed.

I ran into problems keeping track of the virtual interfaces on the hypervisor being used, so I implemented them into the client table on the database. Each client gets assigned a virtual interface for their cluster. This made adding workers to the cluster much more manageable.

Handling the deletion of clients was difficult. I had a bash script that would interface with the proxmox api to power down the cluster, delete the machines and disks, and remove the virtual interface. For the database I decided to keep the client and node data, but just mark it as inactive.

Eventually I was having trouble communicating between the python web application and the four bash scripts. I found a python package called sh. This library allows importing any Linux command into python and handle it as a function.

```
ex.
from sh import ifconfig
print ifconfig("eth0")
```

This meant I could import the libguestfs library and convert all my bash scripts into python. Once I had a class for this, everything was intercommunicating smoothly and not throwing arguments at bash scripts hoping they don’t fail.

Implementing task queues was a big decision. When adding a client and waiting for it to build, what happens if you added another client? What happens if you added multiple workers

at the same time? I found out through testing that you can only make about two clients at the same time before things slow down to a halt. I decided to use Redis as a task queue system and follow the model that GRR itself uses. Each job gets put into a Redis database, and separate worker processes will grab jobs out of the database and complete them. This makes it so the web interface is not blocked while waiting for provisioning to complete. Functions query for status updates on the jobs, so the web interface can display that information to the admin.

I used git to version control and setup my own continuous integration server for unit testing (http://kevinlaw.info/blog/open-source-continuous-deployment/). This means I have a complete history of every save to my code, and that all my code is tested thoroughly. If new code broke some other feature, I was notified immediately. When I was implementing the Redis task queues I created a branch in the repository and hammered away. If it didn't work I could just switch back to the development branch and pretend it never happened.

I learned from GRR that difficult applications to install and configure are not ideal. My application can be installed with a simple script. For continuous integration to work, the application needs to be installed on a fresh operating system quickly and automatically. Since my code was being tested on every save, I made sure from the start that my application met that goal.

Results

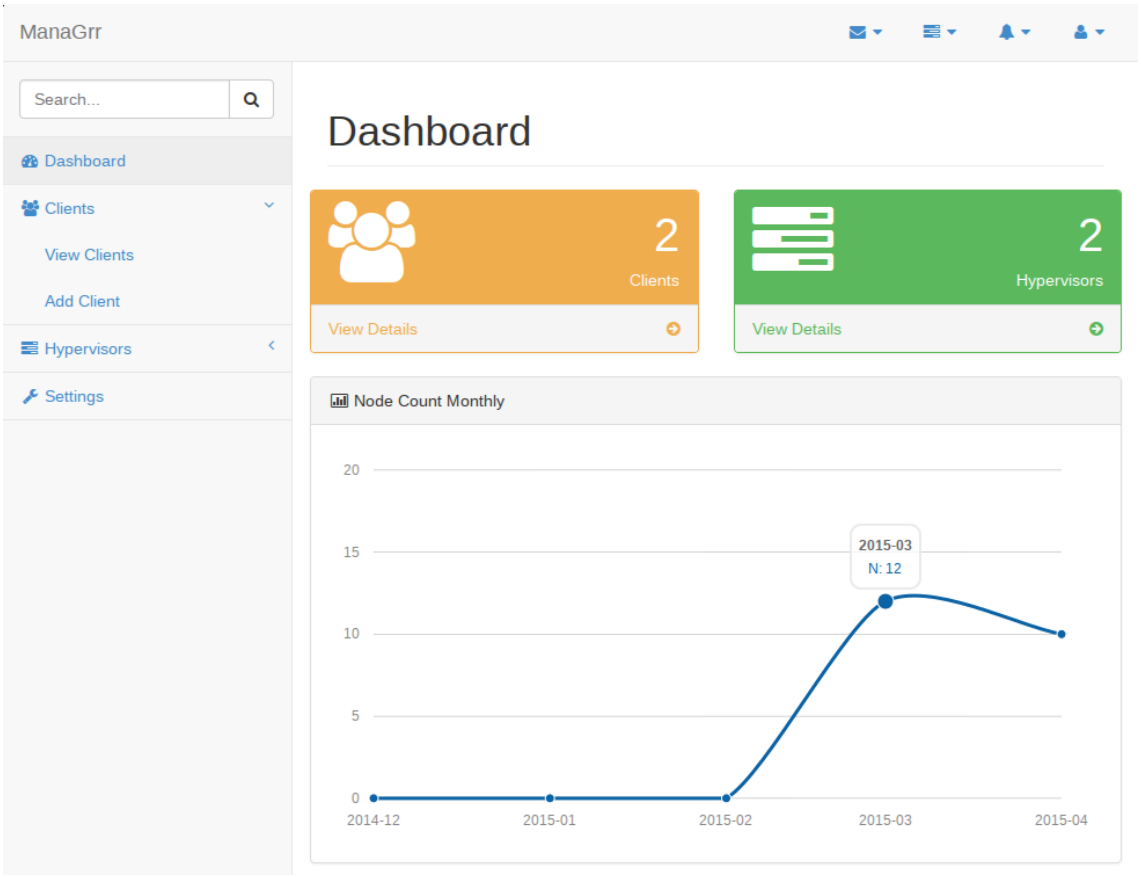
While the project had many obstacles to overcome and some of the original goals were not met, I created a working product to manage and automate creating GRR clusters to sell to clients. "GRR as a Service", if you will.

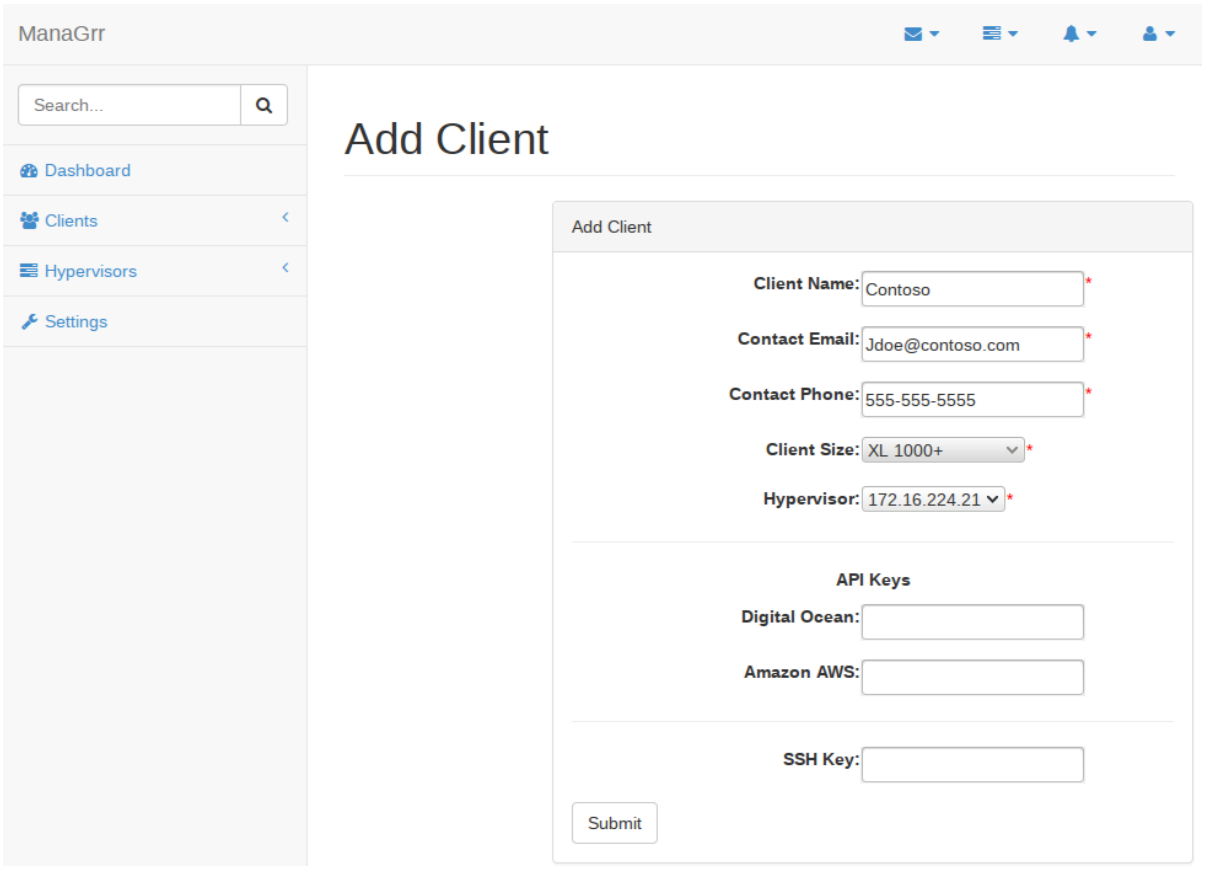
I was not able to find time to provision workers to Digital Ocean or AWS. I have the boilerplate code and database tables to handle it though. I left the code open and documented so anyone can hop in and pick up where I left off.

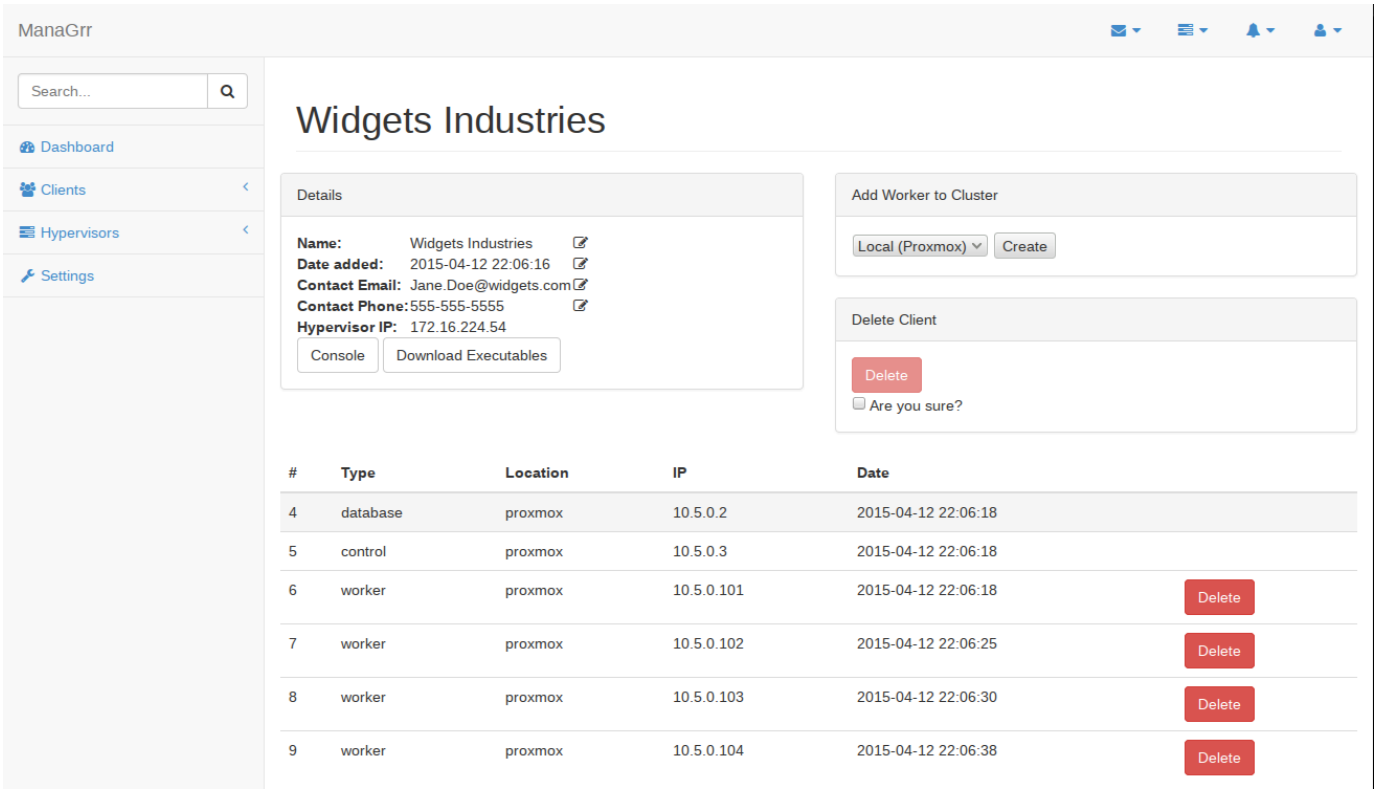
After talking to a GRR developer, he brought up the idea of Dockerizing GRR. Docker is a technology that puts software into containers that can be run anywhere. They are very pluggable and portable. Originally I wanted to use Docker instead of a hypervisor for this project, but there wasn't enough information about Docker out yet, and I wasn't too familiar with how it worked. I realized I didn't have to time to redo my capstone using Docker, so I wrote the code to Dockerize GRR and submitted it to Google https://github.com/google/grr/pull/124.

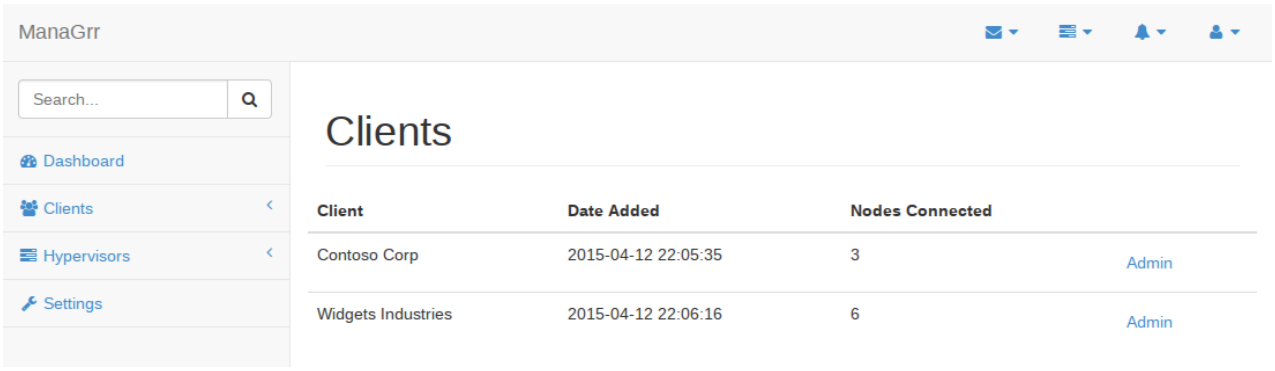
I learned a lot from this project. Tasks that seemed simple and were overlooked turned out to be the hardest. "How do you automate installing an operating system?", A question that I didn't think I had to research too hard to find the answer since in class we are so used to the VMware easy install.

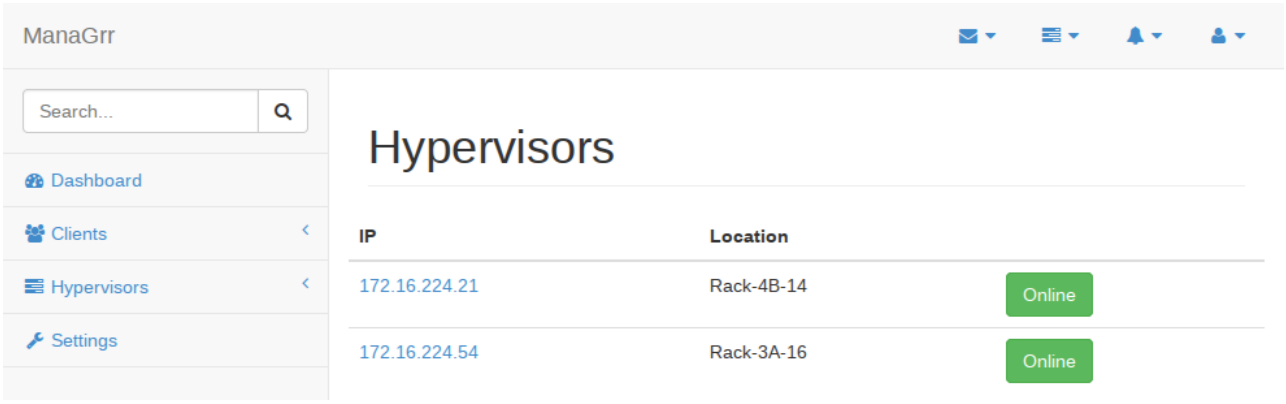
Pictures of ManaGRR




ManaGrr
Search...
Dashboard
Clients
Hypervisors
Settings
Add Client
Add Client
Client Name: Contoso
Contact Email: Jdoe@contoso.com
Contact Phone: 555-555-5555
Client Size: XL 1000+
Hypervisor: 172.16.224.21
API Keys
Digital Ocean:
Amazon AWS:
SSH Key:
Submit


ManaGrr
Search...
Dashboard
Clients
Hypervisors
Settings
Widgets Industries
Details
Name: Widgets Industries
Date added: 2015-04-12 22:06:16
Contact Email: Jane.Doe@widgets.com
Contact Phone: 555-555-5555
Hypervisor IP: 172.16.224.54
Console
Download Executables
Add Worker to Cluster
Local (Proxmox)
Create
Delete Client
Delete
Are you sure?
Type Location IP Date
4 database proxmox 10.5.0.2 2015-04-12 22:06:18
5 control proxmox 10.5.0.3 2015-04-12 22:06:18
6 worker proxmox 10.5.0.101 2015-04-12 22:06:18 Delete
7 worker proxmox 10.5.0.102 2015-04-12 22:06:25 Delete
8 worker proxmox 10.5.0.103 2015-04-12 22:06:30 Delete
9 worker proxmox 10.5.0.104 2015-04-12 22:06:38 Delete


ManaGrr
Search...
Dashboard
Clients
Hypervisors
Settings
Clients
Client
Date Added
Nodes Connected
Contoso Corp
2015-04-12 22:05:35
3
Admin
Widgets Industries
2015-04-12 22:06:16
6
Admin


ManaGrr
Search...
Dashboard
Clients
Hypervisors
Settings
Hypervisors
IP
Location
172.16.224.21
Rack-4B-14
Online
172.16.224.54
Rack-3A-16
Online

References

Carrier, Brian. Open Source Digital Forensics Tools: The Legal Argument . Tech. N.p., 2002. Web. 27 Nov. 2014.

Casey, Eoghan (2004). *Digital Evidence and Computer Crime, Second Edition*. Elsevier. ISBN 0-12-163104-4.

Ellingwood, Justin. "An Introduction to CoreOS System Components." *An Introduction to CoreOS System Components*. DigitalOcean, 5 Sept. 2014. Web. 2 Dec. 2014. <https://www.digitalocean.com/community/tutorials/an-introduction-to-coreos-system-components>.

Charters, Ian. 2009. “The Evolution of Digital Forensics: Civilizing the Cyber Frontier.” <http://www.guerilla-ciso.com/wp-content/uploads/2009/01/the-evolution-of-digital-forensics-ian-charters.pdf>

Cohen, M.i., D. Bilby, and G. Caronni. "Distributed Forensics and Incident Response in the Enterprise." Digital Investigation 8 (2011): S101-110.

“Encase Enterprise” *Guidance Software.* N.p., n.d. Nov 2014 <https://www.guidancesoftware.com/products/Pages/encase-enterprise>

Griffin, Donovan. "Puppet Labs: Advanced IT Automation." *Information Today* 31.5 (2014): 11. *ProQuest.* Web. 29 Nov. 2014.

Hess, Kenneth. "Top 10 Virtualization Technology Companies." *Server Watch* 20 Apr. 2010. Web. 29 Nov. 2014.

GRR-Doc. GRR Documentation Collaboration . N.p., n.d. Nov 2014 <https://github.com/google/grr-doc>

Jaffe, Marley. GRR Capstone Final Paper . Tech. N.p., 2 Dec. 2013. Web. 2 Dec. 2014.

Kharif, Olga, and Ashlee Vance. "Puppet, Chef Ease Transition to Cloud Computing." *Business week* Sep 05 2011: 1. *ProQuest.* Web. 29 Nov. 2014 .

Montgomery, Tommy. “PHP is Dead.” 08 July 2010. Web. 29 Nov. 2014.