

HumanityLink 2.0 aka HLDeuce

vDM Challenge3: Deploy, Manage & Expand



Nigel Hickey; VCIX6-DTM

[@vCenterNerd](https://twitter.com/vCenterNerd)

TABLE OF CONTENTS

| | | |
|---------|---|----|
| 1 | Document Control | 4 |
| 2 | Project Overview | 5 |
| 2.1 | Introduction & Scope | 5 |
| 2.2 | Requirements, Constraints, Assumptions, and Risks | 6 |
| 2.2.1 | Requirements | 6 |
| 2.2.2 | Constraints | 6 |
| 2.2.3 | Assumptions | 6 |
| 2.2.4 | Risks | 6 |
| 3 | Future Site Design | 8 |
| 3.1 | Site Discovery | 8 |
| 3.1.1 | Unmanned Aerial Vehicles (UAV) | 8 |
| 3.1.2 | PX4 | 8 |
| 3.1.3 | QGroundControl | 8 |
| 3.2 | Site Deployment | 10 |
| 3.2.1 | Modular Datacenters | 10 |
| 3.2.2 | Modular Configuration | 10 |
| 3.2.3 | Modular Deployment | 10 |
| 4 | Future Site Management | 12 |
| 4.1 | Site Provisioning | 12 |
| 4.1.1 | Cobbler | 12 |
| 4.1.1.1 | vSphere Management Cluster | 13 |

- 4.1.2 Terraform 15
- 4.1.3 Version Control 19
- 4.2 Operational Procedures 20
 - 4.2.1 GitHub 20
 - 4.2.2 Terraform 21
 - 4.2.3 HumanityLink 27
- 5 References..... 29

1 DOCUMENT CONTROL

Preparation

| Action | Name | Date |
|-------------------|--------------|----------|
| Technical Content | Nigel Hickey | 7/1/2017 |
| Formatting | Nigel Hickey | 7/4/2017 |

Release

| Version | Date Released | Change Notice | Pages Affected | Remarks |
|---------|---------------|---------------|----------------|---------------|
| 1.0 | 7/15/2017 | Internal | All | Initial Draft |
| 1.1 | 7/18/2017 | VDM Release | All | VDM Release |

Distribution

| Name | Organization | Role | E-mail |
|------------------|------------------------|----------------------|---|
| Eric Wright | vDM | Head Canadian, Eh! | eric@discoposse.com |
| Angelo Luciani | vDM | Chief Ginger Officer | Aluciani@gmail.com |
| Melissa Palmer | vDM | Dr. Evil's Sister | vmiss33@gmail.com |
| Byron Schaller | vDM | Judge | byron.schaller@gmail.com |
| Rebecca Fitzhugh | vDM | Judge | rmfitzhugh@gmail.com |
| Lior Kamrat | vDM | Judge | https://twitter.com/LiorKamrat |
| Nigel Hickey | vCenterNerd Consulting | Architect / Engineer | Nigel.Hickey@gmail.com |

2 PROJECT OVERVIEW

2.1 INTRODUCTION & SCOPE

Things have changed...

vCenterNerd Consulting had to do the one thing they never wanted to even think of doing, activating Operation MegaDeath. It was not ideal, but necessary and effective.

Since all datacenter employees who had hands on-site access to facilities at MAC & DNB had to be physically terminated from the earth, our efforts are now even more complex as we scale our environment from three sites to many. The only people left behind after the EatBrains virus are myself and two other trusted engineers who have completed extensive DNA testing for zombie infectious disease as well as completed a comprehensive interrogation after being injected with Candor's Truth Serum.

The three of us must now maintain our current 3 site architecture as well as continue to explore future sites for deployment. We are actively bringing new men & women on board one by one into our new faction after undergoing the same testing as the others. We have located and hired armed security guards from the vDM Season 1 datacenter in Bratislava, Slovakia that was masterminded by the infamous Katarina Wagnerova (VDM004). This new security team will help enforce our new standards going forward, as well as help us find & on-board new datacenter engineers as required.

Our new security team has already began proving their worth in recent weeks. One of the security soldiers who only goes by the name 'Josh01', has recovered what looks to be the remains of a chip used in a T-800 Skynet bot. The chip is said to still contain traces of cybernetic organisms.

Josh01 and his team have been granted approval to continue building their new "robot". The only thing that he kept mentioning over and over was that we must let him design this robot based from the template shown below. He has also dubbed this new cyborg GP-900, something about GingerPower-900 series processors...I am not fully understanding this just yet, but we could use the unit in future missions for sure.

GP-900 cyborg template:



2.2 REQUIREMENTS, CONSTRAINTS, ASSUMPTIONS, AND RISKS

2.2.1 Requirements

| Number | Description |
|--------|--|
| R01 | Centralized remote operations for datacenters |
| R02 | Operations Procedures for managing version control of code |
| R03 | Expansion beyond 3 sites is needed |

2.2.2 Constraints

| Number | Description |
|--------|---|
| C01 | Future sites for new DCs have not yet been declared Zombie-Free-Zones |
| C02 | DC staff has been reduced to a handful after Operation MegaDeath |
| C03 | Design must include currently available products, no futures |

2.2.3 Assumptions

| Number | Description |
|--------|--|
| A01 | FCC does not exist to police drone flight regulations |
| A02 | Modular datacenters are available from vendor |
| A03 | Open Source applications are available for download/use |
| A04 | Robot Army has yet to be fully deployed |
| A05 | WAN connections between datacenters are at least 100mbps & redundant |
| A06 | Deployment Drivers/Pilots have been trained to verify DC components in working order |
| A07 | Cobbler installation & configuration is out of scope for this document |

2.2.4 Risks

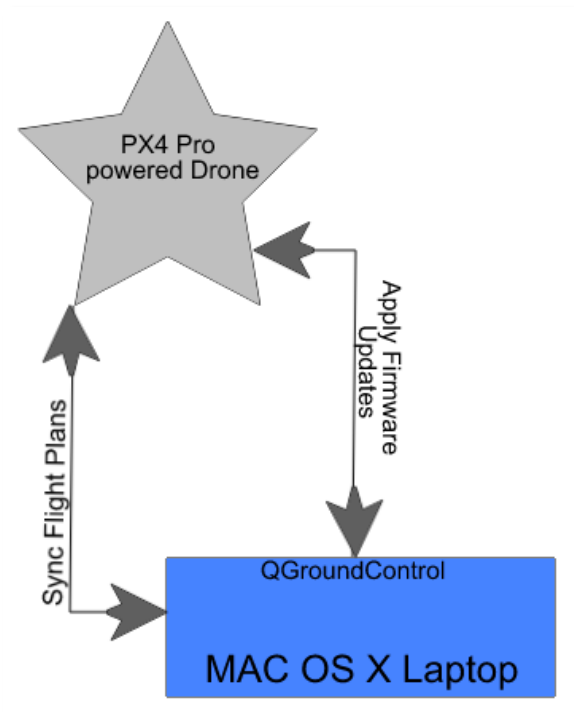
| Number | Description | Mitigation |
|--------|--|---|
| RS01 | New DC deployments in may occur near Zombie infested areas | DC deployments will be modular with minimal human interaction required. |
| RS02 | Off-Earth datacenters are being consider for future growth | Earth DCs will always be considered 1 st , and off-Earth sites only when absolutely needed. |
| R03 | Datacenter Admins may not be able to learn how to code apps or infrastructure without many hours of training | Terraform was chosen as the Infra-as-code application as it does not take much learning of new tech. Direction & training will be minimal |

3 FUTURE SITE DESIGN

3.1 SITE DISCOVERY

3.1.1 Unmanned Aerial Vehicles (UAV)

As we move to expand our datacenter locations, we must be careful as many of these future sites are still considered dangerous and have not been declared zombie free zones. Because of this we are considering the use of Unmanned Aerial Vehicles (UAV) for recon missions to potential new site locations that could still have Zombie activity. These PX4 Pro powered drones will allow us to plan autonomous flight missions to areas of interest. The use of the application QGroundControl will provide full flight control and vehicle setup for PX4 powered UAVs.



3.1.2 PX4

PX4 is platform independent autopilot software (or a software stack/firmware) that can fly Unmanned Aerial Vehicles (UAV). It is loaded (flashed) on vehicle control hardware and together with Ground Control Station it makes a fully autonomous autopilot system.

3.1.3 QGroundControl

The PX4 ground control software is called QGroundControl and is integral part of the PX4 Autopilot System. QGroundControl will run on OS X or Linux systems within our DCs. With the help of QGroundControl you can load (flash) the PX4 on to the vehicle control hardware, you can setup the vehicle, change different parameters, get real-time flight information and create and execute fully autonomous missions.

QGroundControl Key Features:

- Full setup/configuration of PX4 Pro powered vehicles
- Flight support for vehicles running PX4
- Mission planning for autonomous flight
- Flight map display showing vehicle position, flight track, waypoints and vehicle instruments
- Video streaming with instrument display overlays
- Support for managing multiple vehicles
- QGC runs on Windows, OS X, Linux platforms, iOS and Android devices

"Fly View": QGroundControl



Justification

The use of Unmanned Aerial Vehicles (UAV) for recon missions was the best way to get visuals on possible site locations for placing our new datacenters. Using an UAV was easier than losing another member of our already short staffed team. [C01]

3.2 SITE DEPLOYMENT

3.2.1 Modular Datacenters

We have chosen to use Modular Datacenters similar to what IO Datacenters provides (DC in a box) with the D-MODULE (Up to 18 Racks Power – Up to 180kW Dedicated – Cooling, power, fire suppression, security) systems.

D-Module DCs



These modular datacenters can be delivered to our new site locations and be ready to come online once powered up. To assist the delivery & power up of these modular datacenters, the delivery driver or pilot will also carry a payload of human remains (AKA Zombie-Treats) leftover from Operation MegaDeath. Delivery drivers will be instructed to use Zombie-Treats at least 1 mile outside of the intended new datacenter location for distraction purposes while the D-MODULE is positioned & powered up, and connected to uplinks.

Delivery drivers/pilots will have access to two PX4 drones and a Linux laptop running QGroundControl. The drivers/pilots can then leverage the PX4 drone to survey the area prior to physical D-MODULE placement.

3.2.2 Modular Configuration

Prior to these modular datacenters being shipped and at a minimum, vSphere 6.5 will be preinstalled on the 1st three Dell R730's and configured with a Mgmt Cluster to limit the amount of work that needed to be completed after these data centers are powered up on site. The same Networking & Storage used at the MAC & DNB Datacenters will be used within these modular datacenters. Site-to-Site VPN is also part of the pre-configuration.

3.2.3 Modular Deployment

The Operational Procedures for deployment of our modular datacenters is as follows:

1. Deployment driver and/or pilot will drop Zombie-Treats for distraction
2. Deploy modular datacenter to physical site location

3. Verify physical site security with PX4 drone prior bringing online with power connects
4. Verify network connectivity at uplinks
5. Verify Site-Site VPNs between existing datacenters come online with new Modular DC
6. Deployment driver will then contact site admins in MAC & DNB datacenters to verify connectivity to new Modular DC
7. Deployment driver will leave the area immediately after he/she receives confirmations from site admins in MAC or DNB DCs.
8. Deployment driver may leave area ASAP if in danger from Zombies or other threats, but not until at least power has been provided to the Modular DC.

Justification

The use Modular Datacenters will allow us to have a fully operational datacenter “pod” in areas that may still be potentially unsafe from Zombies. These are easily repeatable building blocks that can be scaled up or down as we grow or migrate to different areas of Earth and even possibly Off-Earth locations.

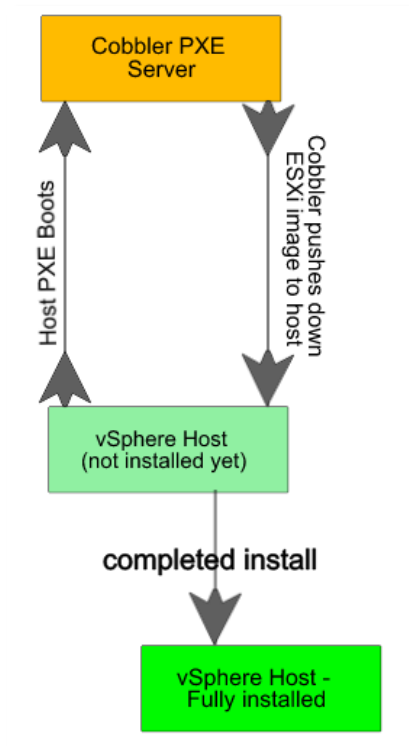
[R03], [A06], [C01], [C02], [RS01], [RS02]

4 FUTURE SITE MANAGEMENT

4.1 SITE PROVISIONING

4.1.1 Cobbler

Cobbler 2.8 will be leveraged to PXE boot any VMware host that was not preinstalled with vSphere for us within our modular datacenters. Cobbler is a Linux installation server that allows for the rapid setup of network installation environments and can help with provisioning, managing DNS and DHCP, package updates, power management, configuration management orchestration, etc.



Cobbler will be installed on a CentOS 6.5 Linux virtual machine within the first vSphere Management cluster that is prebuilt in our modular DC and will serve to be the PXE boot target for deploying additional ESXi hosts. The Cobbler server will reside on a permanent VLAN (500) in order to prevent any production systems from booting from Cobbler by mistake.

| VM | Description | vCPU | RAM | Disk | VLAN |
|------------|-----------------------------|------|-----|------|------|
| MOD-COBB01 | Cobbler PXE Target (CentOS) | 4 | 4GB | 20GB | 500 |
| | | | | | |

4.1.1.1 vSphere Management Cluster

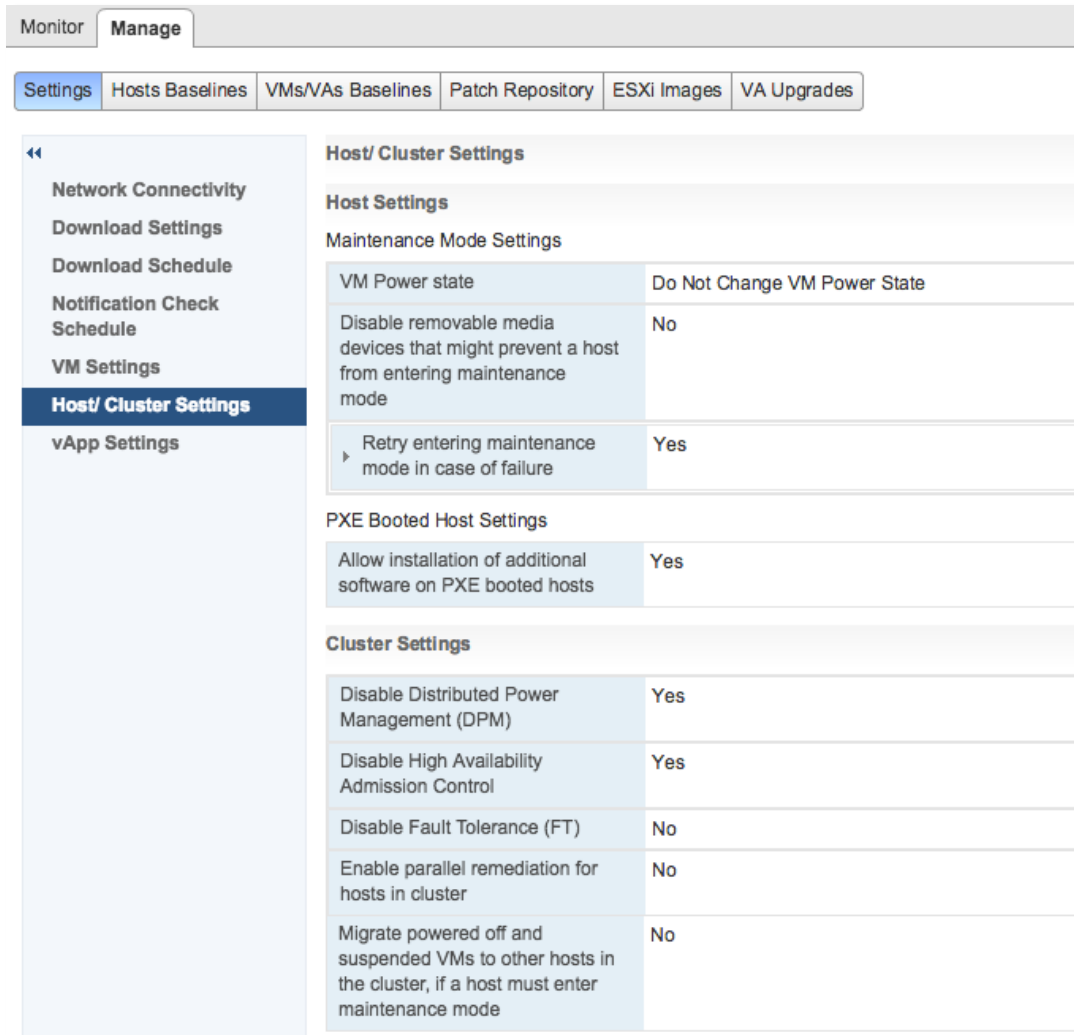
The vSphere Management cluster will contain the same systems setup as our other datacenters in MAC & DNB. These three Dell R730XD servers in our vSphere 6.5 cluster will also leverage VMware Update Manager (VUM) as well as Host Profiles to keep settings consistent as well as have easy access to vSphere & third party software updates, patches, and upgrades.

VUM is setup to download new updates daily at 3AM; seen below in the Download Schedule screen

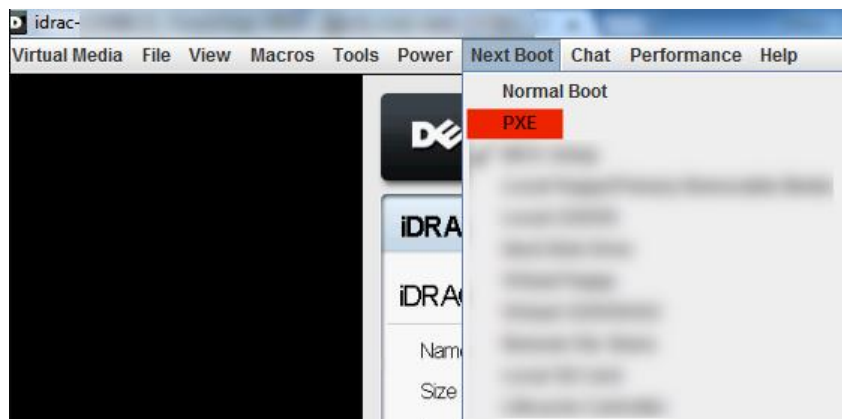
The screenshot shows the VMware vSphere Update Manager (VUM) interface. At the top, there are tabs for 'Monitor' and 'Manage'. Under the 'Manage' tab, there are sub-tabs: 'Settings', 'Hosts Baselines', 'VMs/VAs Baselines', 'Patch Repository', 'ESXi Images', and 'VA Upgrades'. The 'Settings' sub-tab is selected, and on the left, there is a sidebar with options: 'Network Connectivity', 'Download Settings', 'Download Schedule' (which is highlighted), 'Notification Check Schedule', 'VM Settings', 'Host/ Cluster Settings', and 'vApp Settings'. The main area displays the 'Download Schedule' configuration. It includes a table with the following settings:

| Download Schedule | |
|---------------------------|---|
| Enable scheduled download | Yes |
| Task name | VMware vSphere Update Manager Update Download |
| Task description | A predefined scheduled task to download software updates. |
| Frequency | Daily |
| Last run | 7/17/17, 6:03:00 PM CDT |
| Next run | 7/19/17, 3:00:00 AM CDT |
| Send email to | VUM-Admin@vCenterNerd.com |

The following Host settings are also configured in VUM



Additionally, each Dell system outside of the 1st vSphere cluster will be configured via iDRAC to have a "Next Boot" point to PXE. This will allow the future ESXi hosts to boot via PXE and see the Cobbler server to get its config.



Justification

Cobbler was chosen to best help with the automation of ESXi vSphere installations on hosts within each Modular DC. Since we may not be able to have humans at every new Modular DC, this can help satisfy the need for remote management & provisioning. [R01], [A07]

4.1.2 Terraform

Terraform will be leveraged to design and provision our infrastructure virtual machines as well as help manage version control. Installing of infrastructure virtual machines will require terraform configurations as well terraform variables held in .tf files. Terraform can be used for both our vSphere & AWS datacenters if required.

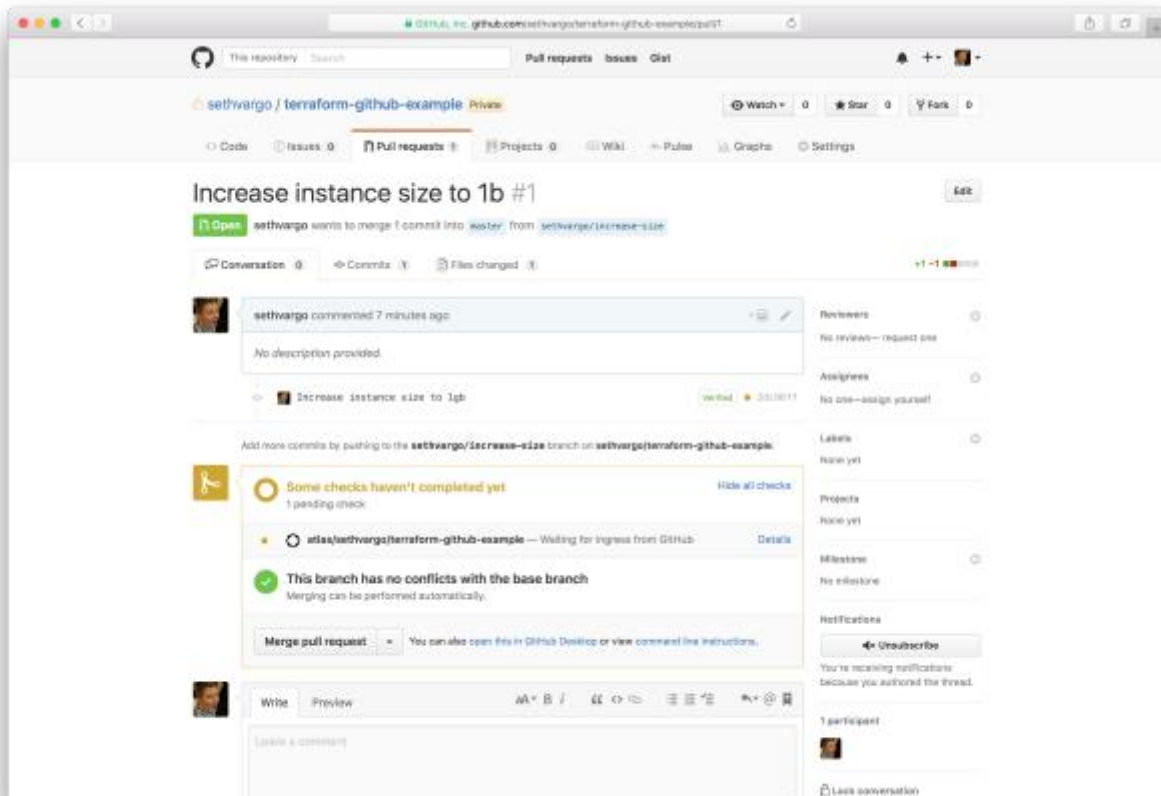
From Terraform's site: *"Terraform enables you to safely and predictably create, change, and improve production infrastructure. It is an open source tool that codifies APIs into declarative configuration files that can be shared amongst team members, treated as code, edited, reviewed, and versioned."*

Terraform is distributed as a binary package for all supported platforms and architectures. Terraform can be installed on Windows, Linux, Solaris, OpenBSD. We will be installing Terraform on a MAC OS X & Linux systems ONLY and saving our Terraform configs (*.tf files) in GitHub for versioning control.

Terraform (Pro version can do this) can connect to GitHub, GitHub can automatically notify Terraform Pro of changes to code at the Version Control Software (VCS) layer. These change notifications, in the form of webhooks, automatically trigger a plan phase. Terraform Pro controls the version of Terraform, the ingress and egress permissions, and securely stores and manages provider credentials. After linking, when new branches are created on this repository, Terraform Pro will automatically execute a dry-run on the configuration changes and report back the results to GitHub through a Pull Request.

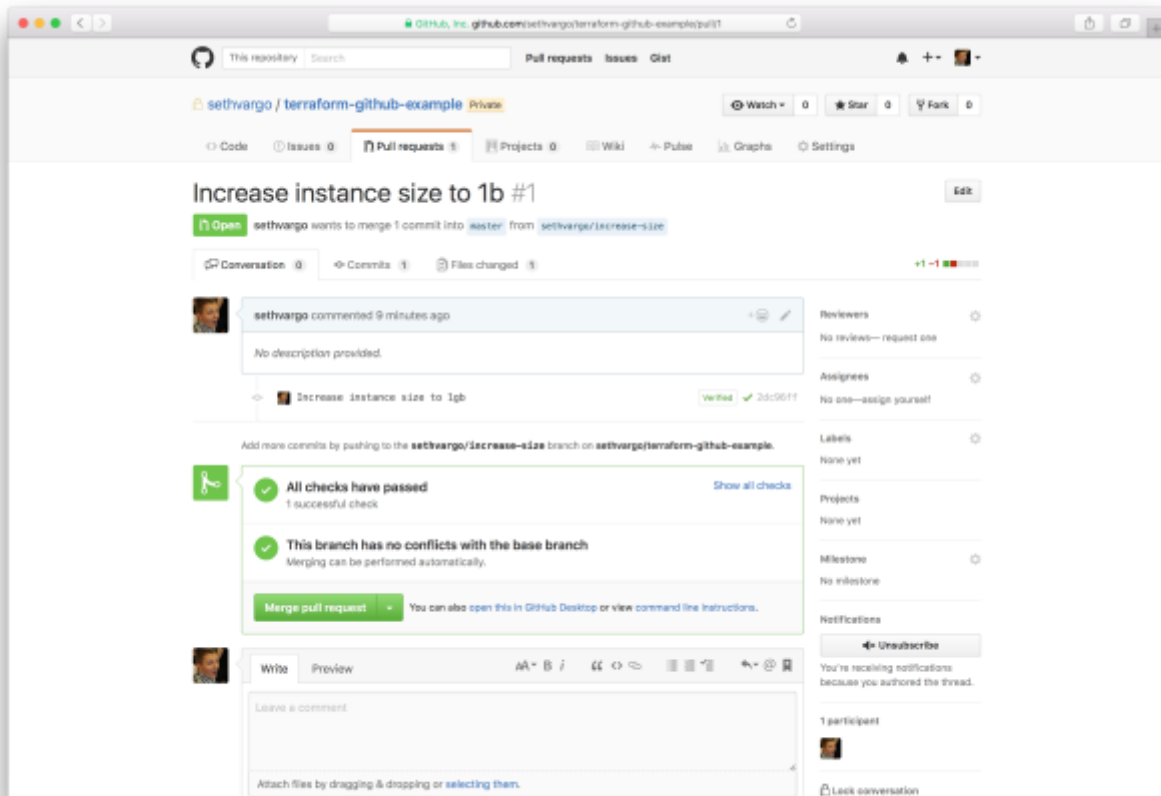
Here is an example below of a GitHub Pull Request that shows Terraform executing the plan phase of changes. Notice the pre-check (terraform plan) & then the Green Checkmark when the changes would apply successfully.

Example: `$ terraform plan`

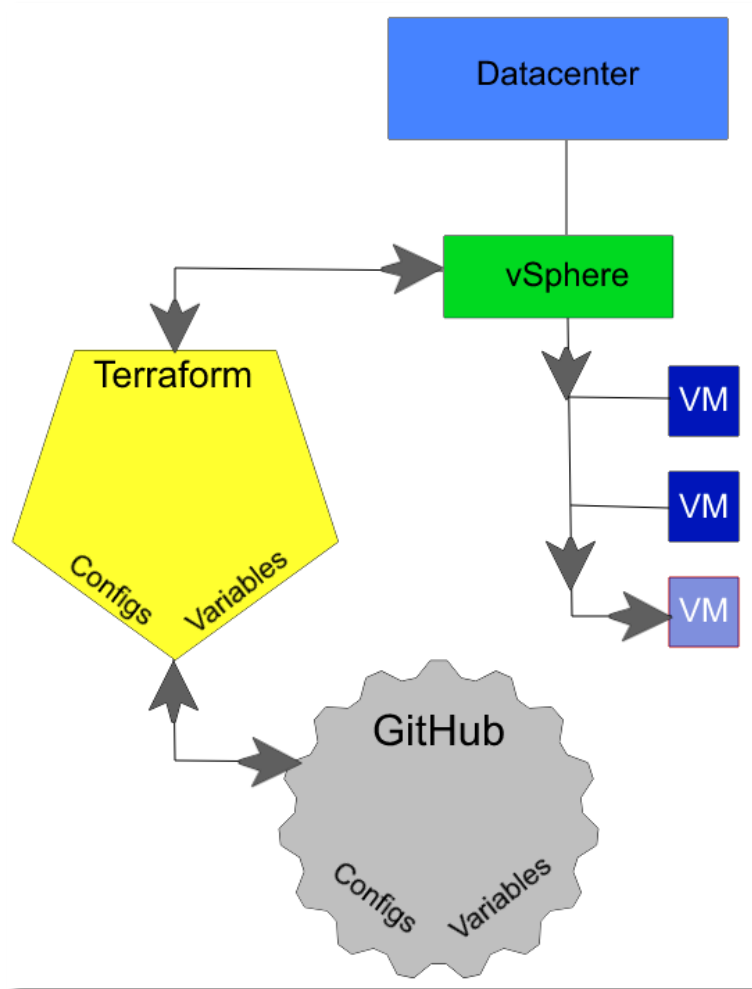


From GitHub you can then jump to Terraform Pro (shown below) to see the output from the Terraform Plan. Once changes are approved from datacenter team members, the code can be merged in GitHub, confirmed in Terraform, then the infrastructure is created or changed as you would/could do manually at the command line.

Example: `$ terraform apply`



Example: Terraform Logical Flow creating vSphere VM



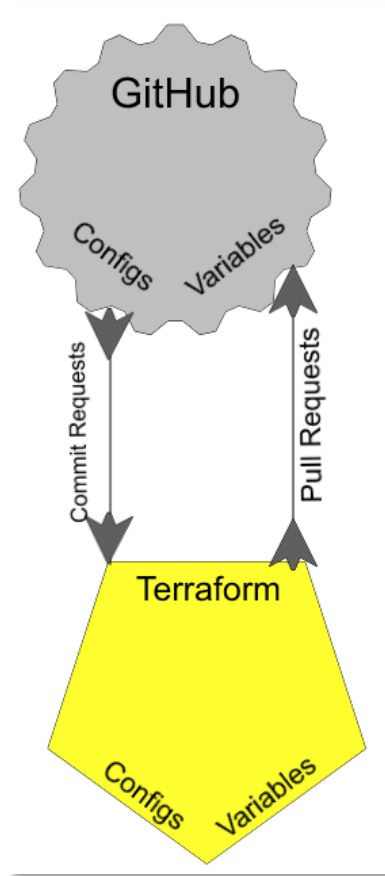
Justification

To have a safe, predictable, and repeatable infrastructure as code deployment model, Terraform Pro was chosen. It is more designed for teams or collaboration than the standard Terraform version which does not scale well. Terraform Pro also allows our teams to meet the requirements of version control by connecting Terraform Pro to GitHub. [R01], [R02]

4.1.3 Version Control

GitHub will be used for maintaining version control of our Infrastructure and Application code. GitHub will also allow our Engineers to work together on code and be remotely dispersed from each other. To hit the ground running, we will leverage the Business version of GitHub because it will allow us to have multiple users in Git,

create Private repositories to share code securely, as well as the Business plan guarantees 99.95% uptime which is always a positive.



Justification

GitHub fit the need for simple setup & configuration of code repositories that can be shared, secured, and versioned without much human effort and learning curve. [R01], [R02], [A03], [C03]

4.2 OPERATIONAL PROCEDURES

4.2.1 GitHub

Following this procedure will enable GitHub to be used for a code repository. Access for all MAC & DNB administrators has been granted already to the vCenterNerd Consulting account. Admins have also been given the proper credentials to login prior to following these steps.

1. Login to our GitHub account for vCenterNerd Consulting at: <https://github.com/>
2. Create a Repository for your datacenter (use MAC for Machrihanish & DNB for Devonport)
 - a. Click the “+” symbol near the upper right corner of the page, and select New Repository
 - b. Name the Repository

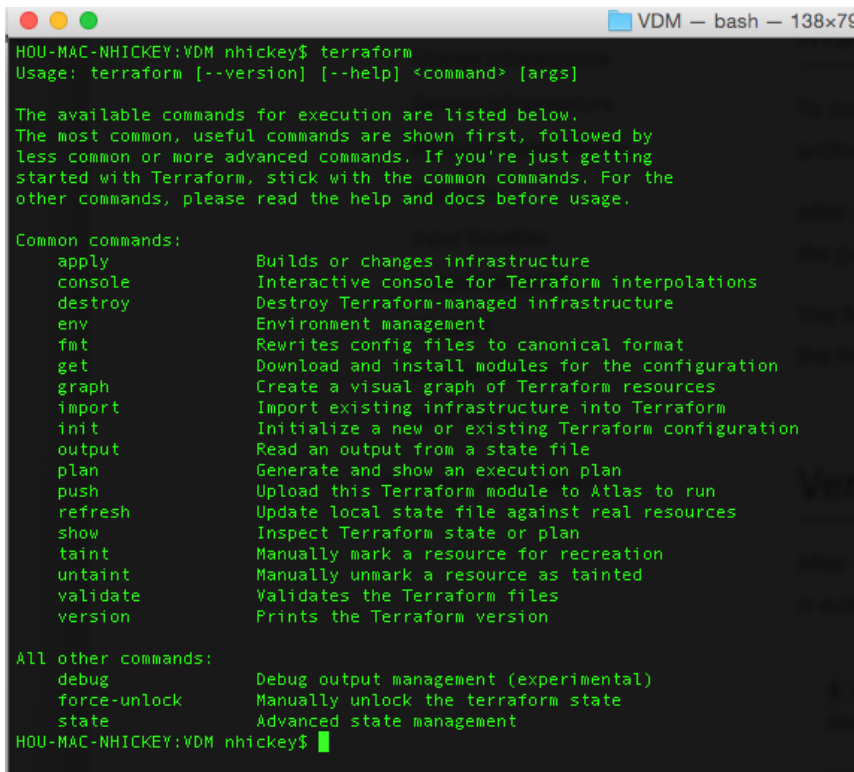
- c. Type a short description (enter DC name & admin who created)
 - d. Choose **Private** as the type
 - e. Select the checkbox **Initialize this repository with a README**
 - f. Click **Create Repository** to finish
3. Next we need to create a Branch within the new Repository you have just made. The Branch will be used when testing or editing before committing the changes to the Master branch
 - a. From your DC named Repository (MAC or DNB), click the drop down list that reads **branch: master**
 - b. In the drop down, begin typing the new Branch name
 - c. When done typing the Branch name, click the blue bar that appears showing **Create branch: "branch-name"** or simply hit **Enter**
4. At this point the needed GitHub setup is completed for the DC admin(s).

4.2.2 Terraform

Following this procedure will enable the use of Terraform on your systems (laptop/desktop). These steps go over Creating, Changing, and Destroying infrastructure via Terraform.

1. Browse to <https://www.terraform.io/downloads.html> to obtain the proper install package for your operating system (OS).
 - a. NOTE: Our datacenters in will only use approved devices to perform Terraform functions. At this time, Mac OS X & Linux (64bit preferred) are approved OS
2. Create a Terraform directory on your OS and save the ZIP file in this location
3. Terraform runs as a single binary named *terraform*, so we need to make sure that the *terraform* binary is available on the PATH of your OS

4. After you have set the PATH to the terraform location in your OS, open a terminal session and run terraform at the command line to verify installation (should look simialr to the example below)

A terminal window titled "VDM - bash - 138x79" showing the output of the 'terraform' command. The prompt is 'HOU-MAC-NHICKEY:VDM nhickey\$'. The output displays the usage, a list of available commands, and a detailed list of common commands with their descriptions. The terminal text is as follows:

```
HOU-MAC-NHICKEY:VDM nhickey$ terraform
Usage: terraform [--version] [--help] <command> [args]

The available commands for execution are listed below.
The most common, useful commands are shown first, followed by
less common or more advanced commands. If you're just getting
started with Terraform, stick with the common commands. For the
other commands, please read the help and docs before usage.

Common commands:
  apply          Builds or changes infrastructure
  console        Interactive console for Terraform interpolations
  destroy        Destroy Terraform-managed infrastructure
  env            Environment management
  fmt            Rewrites config files to canonical format
  get            Download and install modules for the configuration
  graph          Create a visual graph of Terraform resources
  import         Import existing infrastructure into Terraform
  init           Initialize a new or existing Terraform configuration
  output         Read an output from a state file
  plan           Generate and show an execution plan
  push           Upload this Terraform module to Atlas to run
  refresh        Update local state file against real resources
  show           Inspect Terraform state or plan
  taint          Manually mark a resource for recreation
  untaint        Manually unmark a resource as tainted
  validate       Validates the Terraform files
  version        Prints the Terraform version

All other commands:
  debug          Debug output management (experimental)
  force-unlock   Manually unlock the terraform state
  state          Advanced state management

HOU-MAC-NHICKEY:VDM nhickey$
```

5. Next we will need to create our Terraform config file using any plain text editor
6. Open your text editor and create a new file named "vSphere.tf" (all terraform files are named *.tf and they are all loaded when terraform is executed)

7. Use the code example here to build your Terraform config file;

```
~/Desktop/VDM/vSphere_Terra.tf
1  # Configure the VMware vSphere Provider
2  provider "vsphere" {
3      user      = "${var.vsphere_user}"
4      password  = "${var.vsphere_password}"
5      vsphere_server = "${var.vsphere_server}"
6
7      # if you have a self-signed cert
8      allow_unverified_ssl = true
9  }
10
11 # Create a folder
12 resource "vsphere_folder" "HumanityLink" {
13     path = "HumanityLink"
14 }
15
16 # Create a file
17 resource "vsphere_file" "ubuntu_disk_UPLOAD" {
18     datastore = "local"
19     source_file = "/VDM/my_disks/HumanityLink_ubuntu.vmdk"
20     destination_file = "/MAC_DC/disks/HumanityLink_ubuntu.vmdk"
21 }
22
23 # Create a disk image
24 resource "vsphere_virtual_disk" "extraStorage" {
25     size = 2
26     vmdk_path = "myDisk.vmdk"
27     datacenter = "MAC Datacenter"
28     datastore = "VSAN_MAC"
29 }
30
31 # Create a virtual machine within the folder
32 resource "vsphere_virtual_machine" "Web01" {
33     name = "MAC-HLD-WEB01"
34     folder = "${vsphere_folder.HumanityLink.path}"
35     vcpu = 8
36     memory = 16384
37
38     network_interface {
39         label = "VM Network"
40     }
41
42     disk {
43         template = "centos-7"
44     }
45 }
```

8. This is a complete configuration that Terraform is ready to apply and create a vSphere VM.
9. We are also using a config file to hold Variables that Terraform may need
- This file has been loaded in to the vCenterNerd Consulting GitHub acct under the Test-Branch (<https://github.com/vCenterNerd/VDM/tree/Test-Branch>) and should be copied to your local Terraform directory in your OS

10. To test this config run `$ terraform plan` and the output should be similar

```

HOU-MAC-NHICKEY:VDM nhickey$ terraform plan
var.vsphere_password
  Enter a value:
var.vsphere_server
  Enter a value: vc3a

Refreshing Terraform state in-memory prior to plan...
The refreshed state will be used to calculate this plan, but will not be
persisted to local or remote state storage.

The Terraform execution plan has been generated and is shown below.
Resources are shown in alphabetical order for quick scanning. Green resources
will be created (or destroyed and then created if an existing resource
exists), yellow resources are being changed in-place, and red resources
will be destroyed. Cyan entries are data sources to be read.

Note: You didn't specify an "-out" parameter to save this plan, so when
"apply" is called, Terraform can't guarantee this is what will execute.

+ vsphere_file.ubuntu_disk_upload
  datastore: "local"
  destination_file: "/MAC_DC/disk/ HumanityLink_ubuntu.vmdk"
  source_file: "/VDM/sy_disks/ HumanityLink_ubuntu.vmdk"

+ vsphere_folder.HumanityLink
  existing_path: "<computed>"
  path: "HumanityLink"

+ vsphere_virtual_disk.extraStorage
  adapter_type: "ide"
  datacenter: "MAC Datacenter"
  datastore: "VSAN_MAC"
  size: "2"
  type: "eagerZeroedThick"
  vmdk_path: "ayDisk.vmdk"

+ vsphere_virtual_machine.Reb01
  detach_unknown_disks_on_delete: "false"
  disk.#: "1"
  disk.177764683.bootable: ""
  disk.177764683.controller_type: "scsi"
  disk.177764683.datastore: ""
  disk.177764683.lops: ""
  disk.177764683.keep_on_remove: ""
  disk.177764683.key: "<computed>"
  disk.177764683.name: ""
  disk.177764683.size: ""
  disk.177764683.template: "centos-7"
  disk.177764683.type: "eager_zeroed"
  disk.177764683.uuid: "<computed>"
  disk.177764683.vmdk: ""
  domain: "vsphere.local"
  enable_disk_uuid: "false"
  folder: "HumanityLink"
  linked_clone: "false"
  memory: "16384"
  memory_reservation: "8"
  soid: "<computed>"
  name: "MAC-HLD-VEB01"
  network_interface.#: "1"
  network_interface.0.ip_address: "<computed>"
  network_interface.0.ipv4_address: "<computed>"
  network_interface.0.ipv4_gateway: "<computed>"
  network_interface.0.ipv4_prefix_length: "<computed>"
  network_interface.0.ipv6_address: "<computed>"
  network_interface.0.ipv6_gateway: "<computed>"
  network_interface.0.ipv6_prefix_length: "<computed>"
  network_interface.0.label: "VM Network"
  network_interface.0.mac_address: "<computed>"
  network_interface.0.subnet_mask: "<computed>"
  skip_customization: "false"
  time_zone: "Etc/UTC"
  uuid: "<computed>"
  vcpu: "8"

Plan: 4 to add, 0 to change, 0 to destroy.
HOU-MAC-NHICKEY:VDM nhickey$

```

11. The last steps are to setup the OAuth settings for GitHub to access Terraform
- Access <https://github.com/settings/developers>
 - Click **Register a new application** to begin this process
 - Follow the steps here with those given in the email you have received from vCenterNerd consulting that includes; Homepage URL, App descriptions, and Authorization Callback URL.
12. Setup of Terraform & GitHub integration is complete

13. These steps can also be followed for creating a .tf file for adding/changing/editing infrastructure at our AWS or MAC & DNB datacenters. If this was for AWS you would need to obtain the AWS access key and secret key, available from your DC site Admin upon request.

a. An AWS terraform config would look similar to this:

```
provider "aws" {  
  access_key = "ACCESS_KEY_HERE"  
  secret_key = "SECRET_KEY_HERE"  
  region     = "us-east-1"  
}  
  
resource "aws_instance" "example" {  
  ami           = "ami-2757f631"  
  instance_type = "t2.micro"  
}
```

b. Saving this code to an *AWS.tf* file and then placing it in your Terraform directory is all that is needed to utilize it.

Changing Infrastructure

The next steps will be for Changing infrastructure with Terraform. As you change Terraform configurations, Terraform builds an execution plan that only modifies what is necessary to reach your desired state. Preforming changes this way allows for version control of your configurations and also your desired state so you can see how the infrastructure evolved over time.

1. If we use the sample code from Step 13a in the previous section, we will focus on editing the AMI reference to change from an Ubuntu 16.04 LTS AMI to an Ubuntu 16.10 AMI.

```
provider "aws" {  
  access_key = "ACCESS_KEY_HERE"  
  secret_key = "SECRET_KEY_HERE"  
  region     = "us-east-1"  
}  
  
resource "aws_instance" "example" {  
  ami           = "ami-2757f631"  
  instance_type = "t2.micro"  
}
```

a.

```
resource "aws_instance" "example" {  
  ami           = "ami-b374d5a5"  
  instance_type = "t2.micro"  
}
```

b.

2. Next run *\$ terraform plan* to see the changes that could be applied

```
$ terraform plan
# ...

-/+ aws_instance.example
  ami: "ami-2757f631" => "ami-b374d5a5" (forces new resource)
  availability_zone: "us-east-1a" => "<computed>"
  ebs_block_device.#: "0" => "<computed>"
  ephemeral_block_device.#: "0" => "<computed>"
  instance_state: "running" => "<computed>"
  instance_type: "t2.micro" => "t2.micro"
```

- a.
 - b. The prefix "-/+" means that Terraform will destroy and recreate the resource, versus updating it in-place. While some attributes can do in-place updates (which are shown with a "~" prefix), changing an AMI on EC2 instance requires a new resource altogether.
3. The last step here would be to push the changes via `$ terraform apply`

```
$ terraform apply
aws_instance.example: Refreshing state... (ID: i-64c268fe)
aws_instance.example: Destroying...
aws_instance.example: Destruction complete
aws_instance.example: Creating...
  ami: " " => "ami-b374d5a5"
  availability_zone: " " => "<computed>"
```

a.

Destroying Infrastructure

The last procedure is to be only used when all datacenter admins have been first notified. These steps are samples, and real production code changes will be managed via GitHub with proper approvals. The purpose here is to demonstrate the simplicity of changing and destroying environment via terraform.

1. Destroying sites or infrastructure may be required in extreme situations to satisfy needs during Zombie or Rouge intruders to our datacenters
2. Run `$ terraform plan -destroy` to see how the plan will work 1st (the "-" indicates removal)

```
$ terraform plan -destroy
# ...

- aws_instance.example
```

a.

3. Run `$ terraform destroy` to do just that, destroy your infrastructure

```
$ terraform destroy
aws_instance.example: Destroying...

Apply complete! Resources: 0 added, 0 changed, 1 destroyed.

# ...
```

a.

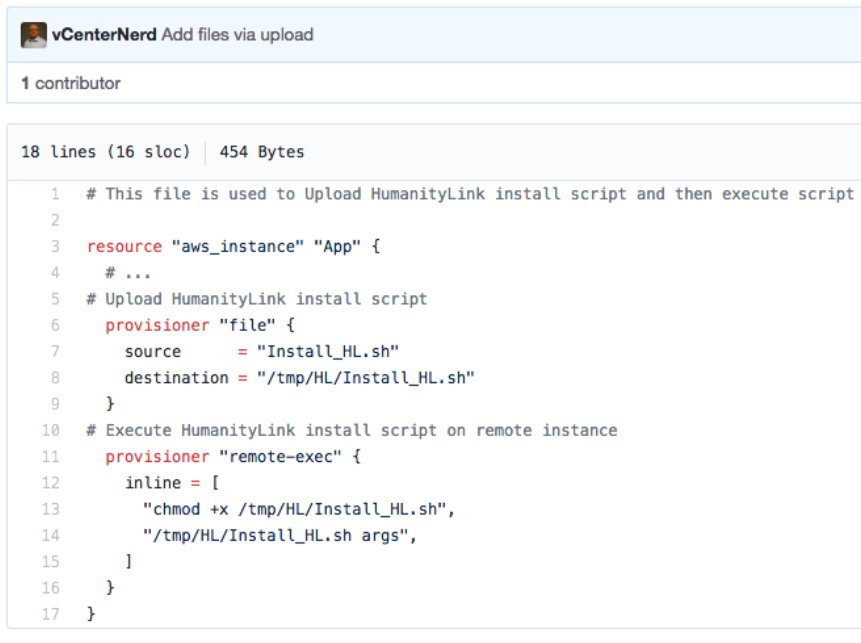
4. The terraform `destroy` command should ask you to verify that you really want to destroy the infrastructure. Terraform only accepts the literal "yes" as an answer as a safety mechanism. Once entered, Terraform will go through and destroy the infrastructure.

4.2.3 HumanityLink

Since we are leveraging Terraform for infrastructure code, we will also use it for as much application means as we know how to. Terraform Provisioners will be used to execute scripts on the remote instances we create in our datacenters. The examples below are referencing AWS resources, but can also be used to run scripts in out vSphere environments also.

Installing HL on an AWS instance:

1. Create a new .tf file called "HL.tf" can save it to the Terraform directory used in previous steps
2. Edit the new HL.tf file to include the below code snippet:




The screenshot shows a code editor window titled "vCenterNerd Add files via upload". It displays a Terraform configuration file with 18 lines of code. The code defines an AWS instance resource named "App" and includes two provisioners: "file" to upload the HumanityLink install script and "remote-exec" to execute the script on the remote instance.

```
18 lines (16 sloc) | 454 Bytes
1  # This file is used to Upload HumanityLink install script and then execute script
2
3  resource "aws_instance" "App" {
4    # ...
5    # Upload HumanityLink install script
6    provisioner "file" {
7      source      = "Install_HL.sh"
8      destination = "/tmp/HL/Install_HL.sh"
9    }
10   # Execute HumanityLink install script on remote instance
11   provisioner "remote-exec" {
12     inline = [
13       "chmod +x /tmp/HL/Install_HL.sh",
14       "/tmp/HL/Install_HL.sh args",
15     ]
16   }
17 }
```

3. The code saved in this file should be added to your main terraform config that deploys infrastructure so that once the instance is stood up, the scripts you wish to run will kick off after terraform has built the instance(s).

- a. Example of this Provisioner within a full AWS terraform config file:

 vCenterNerd Add files via upload

1 contributor

23 lines (22 sloc) | 544 Bytes

```
1 provider "aws" {
2   access_key = "ACCESS_KEY_HERE"
3   secret_key = "ACCESS_KEY_HERE"
4   region     = "eu-west-1"
5 }
6 resource "aws_instance" "App" {
7   ami           = "ami-2757f631"
8   instance_type = "t2.micro"
9
10  # ...
11  # Upload HumanityLink install script
12  provisioner "file" {
13    source      = "Install_HL.sh"
14    destination = "/tmp/HL/Install_HL.sh"
15  }
16  # Execute HumanityLink install script on remote instance
17  provisioner "remote-exec" {
18    inline = [
19      "chmod +x /tmp/HL/Install_HL.sh",
20      "/tmp/HL/Install_HL.sh args",
21    ]
22  }
23 }
```

5 REFERENCES

Version-Controlled Infrastructure with GitHub & Terraform

<https://www.hashicorp.com/blog/version-controlled-infrastructure-with-github-and-terraform/>

Terraform Install

<https://www.terraform.io/intro/getting-started/install.html>

Terraform Providers (AWS & vSphere)

<https://www.terraform.io/docs/providers/aws/index.html>

<https://www.terraform.io/docs/providers/vsphere/index.html>

Managing GitHub with Terraform

<https://www.hashicorp.com/blog/managing-github-with-terraform/>

Cobbler Manual - Version 2.8.x

<http://cobbler.github.io/manuals/2.8.0/>

Using Cobbler to Rapidly Deploy ESXi With Zero Touch Configuration

<http://www.nerdknobs.net/using-cobbler-to-rapidly-deploy-esxi-with-zero-touch-configuration-2/>

Dronecode

<https://www.dronecode.org/documentation/>

PX4 Autopilot User Guide

<https://docs.px4.io/en/>

QGroundControl User Guide

<https://www.gitbook.com/book/donlakeflyer/qgroundcontrol-user-guide/details>

vCenterNerd GitHub

<https://github.com/vCenterNerd>