How to Tame Your Clouds with Automation

September 21, 2023
Contents

• Introductions
• Why automation
• What tools and how they're used
• Demo
Disclaimer

Despite our attempts to keep this high level, there are parts in this presentation where we do get technical.
Who are we?

• University of Florida IT (UFIT)
  • Infrastructure and Communication Technology Team (ICT)
    • Hyperconverged Infrastructure Team (HCI)
      • Cloud Enablement Team (CE)
Members

• Directors
  • Saira Hasnain – Associate VP and Deputy CIO
  • Barry Kinter – Associate Director for Hyperconverged Infrastructure Team

• Cloud Enablement Team
  • Eli Ben-Shoshan – Pre-Eminent Systems Administrator
  • Nicholas Cecere – Systems Administrator 5
  • Keith Sanders - Systems Administrator 5
  • Paul Smith - Systems Administrator 5
  • Eli Meister - Systems Administrator 5
  • Derek Gales - Systems Administrator 5
What does UFIT do?

- University of Florida has a distributed IT model
- Each college/department has some local IT
- UFIT is the overall central IT department responsible for IT direction of the University as a whole and Enterprise IT Services
- Plays a key part in advancing student success via the use of technology
- We operate like a public utility, providing shared Information Technology services throughout the University community
- Our Products are common: Infrastructure, Operations, Systems and Support
What does ICT do?

• Infrastructure
  • All of Campus networking
    • From the building to Internet connectivity
  • Private Cloud deployed on VMWare
  • Public Cloud access
  • Storage Services for Campus
  • Two Data centers in Gainesville
What does ICT do?

- Operations
  - Acts as Network Operations Center (NOC)
  - Operation Staff 24/7 responding to service alerts and customer calls
What does ICT do?

• Systems
  • Authentication and Identity Management
    • Active Directory for campus
    • SAML2 via Shibboleth
  • Infrastructure components for campus ERP system
  • Manage cloud services
    • Office 365
    • Google Workspaces
    • Dropbox
    • Zoom
What does ICT do?

• Support
  • 3rd tier in support for all services provided
  • Help Desk is the 1st Tier
  • 2nd tier is usually handled by local IT
What does ICT do?

• Offer these Services to both Enterprise customers and to hosting customers across campus
Private Cloud Infrastructure

• Two Data Centers in 5 miles apart
• ~100 ESXi hosts
• >3500 VMs
• Synchronously replicated storage via NetApp Metrocluster
• Resilient architecture designed with 2 availability zones
• Each zone has separate
  • Compute
  • Storage
  • Networking
Private Cloud Offerings

• **Infrastructure as a Service**
  • VMs available for self-service by hosting customers around campus
  • Enterprise customers can self-service in the same way or use automations we will talk about today

• **Platform as a Service**
  • Databases
  • File Shares
  • Web Hosting
• At our scale, Automation is a must for:
  • Consistency of deployments
  • Pace of requests is only increasing
  • Management and Campus priority inversion
Terraform + Ansible = Terrable but Awesome!
Terraform

- **Infrastructure as Code (IaC) tool**
  - Lots of providers that so that we can manage most any infrastructure

- **Uses desired state to deploy infrastructure**
  - You tell it the final state (with some hints) and it will try and get there
  - No need to tell it each step in the process
  - Will create a dependency graph which it compares to the current, desired, and last known state and will generate the steps to get to your desired state

- **Modular (more on this later)**
  - Let's you create reusable versioned modules with clean interfaces so that you can accomplish the same task the same way over and over
Terraform

• Really good at managing and deploying base components for a system

• Examples:
  • Deploy a VM
  • Manage day 2 operations like add a disk to an existing VM or change networking
  • Manage DNS entries
  • Manage DHCP reservations
Terraform

- Needs to store current state someplace
- Not so good at managing attributes within an OS deployment like:
  - Password maps
  - Software installs
  - Networking configuration
- We have another tool for that......
• Show some terraform code
Ansible

• Ansible is also an IaC tool
• Does not use desired state but instead uses a procedural approach via playbooks
  • You give it the steps to run and it will take them for you
• Uses an inventory file to know what hosts to connect to with specified credentials
Ansible

• Fantastic cross OS support
  • Windows
  • Linux

• Excellent for:
  • Managing Users
  • Deploying and configuring software
  • Creating filesystems and mount points
  • Managing Docker containers

• Modular (more about this later)
  • Create reusable components called roles
Ansible

- Could be used to deploy core infrastructure components like VMs but we think the desired state in Terraform is a better fit
  - Terraform detects, notifies, and can remediate drift
  - Ansible can't really detect drift
  - Admin needs to account for drift when writing playbooks
• Show a simple Ansible playbook
Use cases

- We use Terraform to:
  - Deploy a VM from a template
  - Set its cloud-init so that it has initial networking
  - Register the VMs in DNS via Infoblox
  - Create the Ansible inventory file

- We use Ansible to:
  - Apply updates
  - Install software
  - Setup extra filesystems and mount points
  - Configure software
Bonus - Vagrant

• Where do the VM templates come from?
• Hashicorp Vagrant
• Vagrant is a tool used to create a virtualized environments
• Can create:
  • VMware VMs
  • VMware templates
  • AWS AMI
  • Azure VM Image
• You can call Ansible playbooks during a Vagrant run
Secrets

• What are they?
  • Password
  • TLS Private Keys
  • API access tokens

• We all have them and need a secure way to get them onto systems
• We might need to restrict which teams can see which password
• We all "should" be rotating our secrets on a regular basis, right?
  • Required by many security standards (FedRAMP Moderate, PCI)
• We need to have an inventory of which systems have which secrets in case "something" happens
• Hashicorp Vault is a secrets management engine

• Can store static secrets:
  • Username/Password
  • Private Keys

• Can interact with authentication systems to generate and vend dynamic secrets
  • Active Directory
  • AWS IAM
  • Azure AD

• Built-in policy engine so that you can limit how secrets are shared
Vault

• Highly Available
  • Uses raft protocol to replicate secrets amongst multiple nodes

• Encrypts secrets at rest
  • Well tested and hardened system of encryption for secrets at rest

• Can authenticate with lots of authentication backends
  • LDAP
  • AWS IAM
  • Kerberos
  • Azure AD
  • JWT

• UF has Vault authenticate using LDAP to the Duo LDAP proxy which gives us 2-factor for secrets

• Also gives us LDAP groups to identity which users are in which teams
Vault

- Vault has an agent that can be deployed on a VM
- Agent will check in with Vault on a regular basis to see if a secret has changed
- Agent has a templating system (based on Go template) that can replace/rewrite a file if a secret has changed
- Agent can call a script before and/or after a secret change
- UF uses the vault agent to update TLS private keys automatically
- But how do you authenticate a system to Vault?
• Vault has a concept of a Role
• A role connects:
  • Backend Authentication System
  • Policies
• This is how you map a set of authenticated entities to policies which is where you limit access to certain secrets
Vault

• Vault has a powerful abstraction called an AppRole
• Instead of using only a token (like an access key or username/password) to identify an authenticating entity, it uses a Role ID and a Secret ID
• The Role ID is the Role that the entity would like to assume
  • It is not sensitive
• The Secret ID is a secret token used to authenticate to Vault
  • Once it authenticates it uses a session token from there on out to keep access
• The combination of the two gives an entity access to a set of secrets based on the policy assigned to the role
• Vault Agent uses an AppRole to authenticate with Vault
• Ansible interacts with Vault during VM deployment to request the Secret ID on behalf of the VM and places the Secret ID in a file on the VM
1. Mount AppRole auth backend
2. Create policy and role for app
3. Get Role ID
4. Embed Role ID to the machine image
5. Provision
6. Get wrapped Secret ID
7. Return wrapping token
8. Deliver wrapping token
9. Unwrap Secret ID
10. Login with Role ID & Secret ID
11. Return client token
Quick Summary

• Terraform is used to deploy the infrastructure components
• Ansible is used to configure any operating systems
• Vault is used to manage secrets
• Terraform and Ansible interact with Vault to gain access to secrets
• So what orchestrates this seemingly complicated dance

GitLab is the glue that ties all these components together!
• GitLab is a lot more than just a fancy web interface for managing git repos
  • Issue Tracking
  • Merge Request with approvals
  • Deployment pipelines triggered based on various events
  • Terraform
    • Registry for modules
    • State repository
  • Container registry
• Deployment Pipelines are what facilitate Continuous Integration / Continuous Deployment (CI/CD)
• Pipelines specify a set of stages
• Each stage has a set of steps that will be executed in order
• Dependencies can be setup between stages so that some can run in parallel and others run in series
• Pipelines are modular in such that you can import stages from another pipeline
• GitLab allows for the creation of centrally managed stages that can be imported for use by other pipelines
• This reuse of centrally managed stages is what allows for standards to be defined and used by all
• All these pipelines run on runners which could be:
  • Docker based
  • Kubernetes based
  • Host with an agent
• We have different runners deployed for different groups
  • Usually this is because the end point that is being manipulated via CI is in a restricted network
• Most of our pipelines need to have access to some secrets
  • vSphere username/password to create VM
  • AWS IAM access key id and secret

• Gitlab has a JWT token that it uses to authenticate to Vault

• When Gitlab authenticates to Vault it adds claims to the token which specify which project or groups it is acting on behalf of

• Vault maps the claims to the appropriate Role

• The Role then enforces a set of policies

• The policies specify which secrets can be accessed
• This might seem complicated and overly complex but we can assure you it is not once you get used to all the components.

• Don't expect to accomplish this overnight. It takes a while to get to this level of maturity. I took us about a year to get here and to be honest there are still some teams that have not really gotten onboard.