RedTeamOps

Mert Coskuner
Caglar Cakici

Agenda

- Introduction
  - Who we are
  - What we do
- Red Team
  - Red Team Lifecycle
  - Models
  - OpSec
  - Infrastructure design
- DevOps
  - DevOps practices
  - Why automation?
  - Why Terraform?
- Red Team meets DevOps: Example
- Maintenance

Who we are

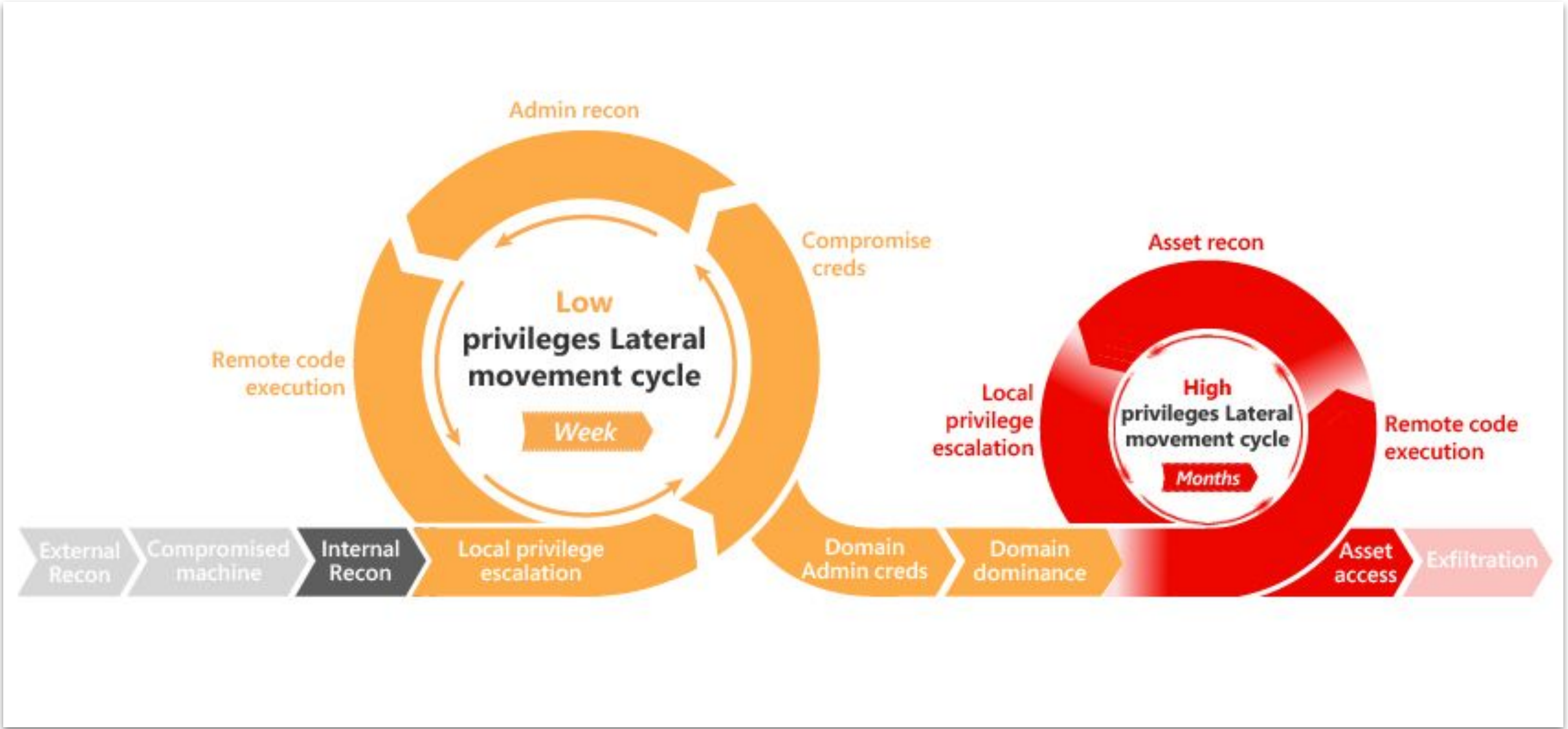
- Security Engineer(s) at Trendyol
- Red team member(s) at Hitcat
- Member(s) of non-profit cyber security organisation BlackBox Security
- Blog at @trendyoltech
- Perform red team and 0-day research

What we do

- Plan an engagement
- Design and automate infrastructure
- Create pipeline for change management
- Iterate over the design

Red Team

Red Team Lifecycle



Operation Models

- Full scope penetration tests
- Long-term red team operations
- War games
- Adversary simulation

Full Scope Penetration Tests

- Attempt to gain a foothold into target, elevate rights and steal data
- Mimics the targeted attack process an actor executes
- Provides data point about the state of a security program
- Needs to be given time and resources

Long-term Red Team Operations

- Gain, elevate and maintain access to different units over long period time
- Allow assessor to work towards perfect knowledge that long-term embedded adversary would have which includes;
  - Network map,
  - Key individuals to target,
  - Valuable user activity within the network
- Besides perfect knowledge, it offers insights to what it takes to keep and maintain access over long periods of time to the network

War Games

- Stage red vs blue war games to train and evaluate defense staff
- Depends on organizer’s needs, each event is different and have different goals
- Some events compress a multi-year scenario into days or weeks
- Provides a safe opportunity to exercise processes and team roles in a fast-paced setting
- Provides blue team to observe and adapt to an adversary thinking

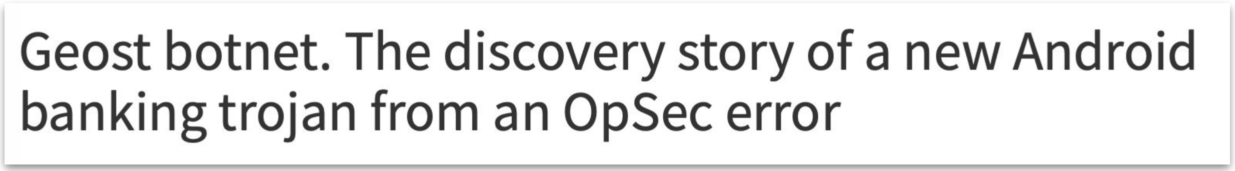
Adversary Simulation

- Exercise a scenario, mock-up or real one in a realistic timeline
- Goal is to generate realistic observable activity for each part of the timeline
- Each executed events are a discussion point for later
- These events are used to validate procedures which is also an opportunity to identify procedure and technology gaps in the organization

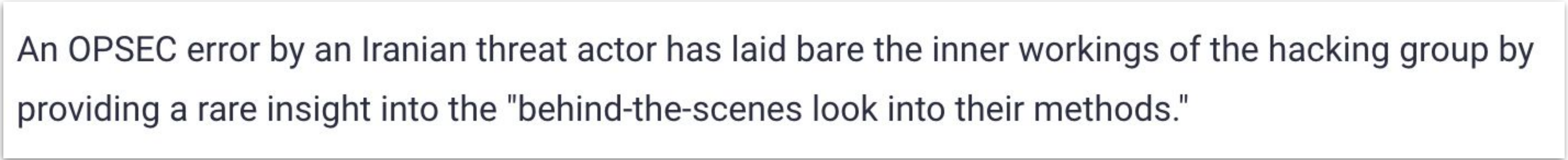
Operations Security (OpSec)

- OpSec is, a military term, an analytical process used to deny an adversary information that could compromise the secrecy and/or the operational security of a mission
- It started as a military process, but over the years it has transformed into something bigger

“... the discovery of the Geost botnet was possible because of several OpSec mistakes, including the use of the HtBot illegal proxy network, **not encrypting their command-and-control servers**, **re-using security services**, trusting other attackers with less OpSec, and **not encrypting their chat sessions** …”


Geost botnet. The discovery story of a new Android banking trojan from an OpSec error

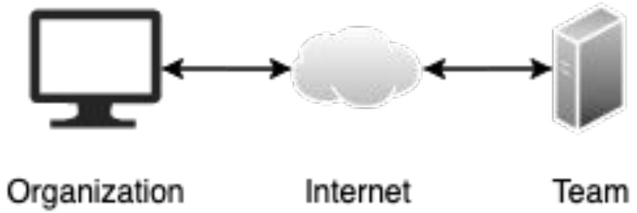
“... Forceful made a series of mistakes that gave up the command and control (C2) details of botnets used to carry out the purchased DDoS attacks in a separate discussion about file encrypting malware.”


An OPSEC error by an Iranian threat actor has laid bare the inner workings of the hacking group by providing a rare insight into the "behind-the-scenes look into their methods."

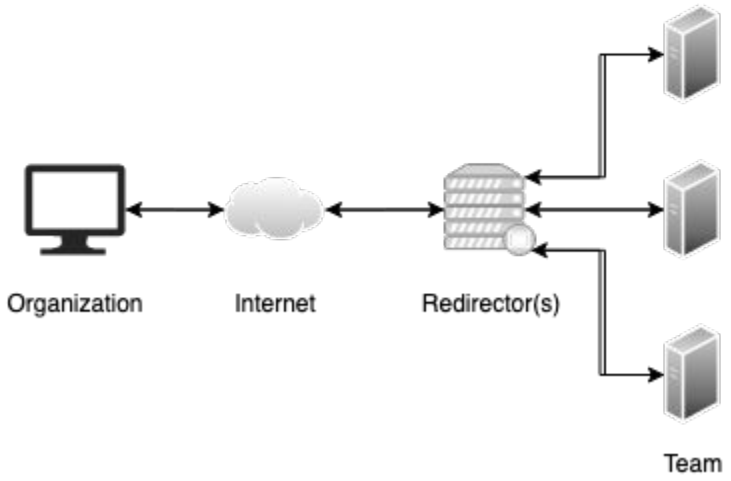
“The IBM researchers said they found the videos on a virtual private cloud server that was left exposed due to a misconfiguration of security settings. The server, which was also found to host several ITG18 domains earlier this year, held more than 40 gigabytes of data.”

Infrastructure Design

Standard penetration testing setup

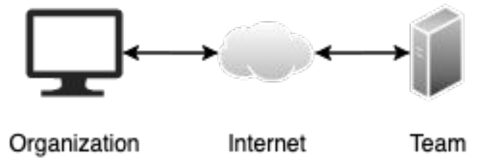


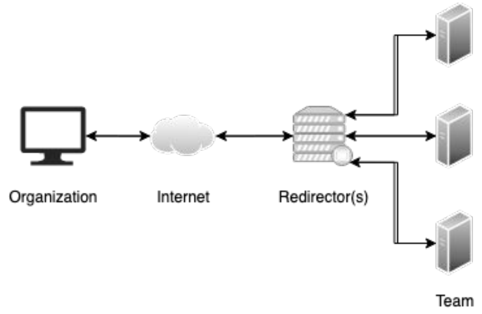
Simple red teaming setup



Differences

- Blends in
- Easy to build defenses against blue team
- Easy to recover against defensive measures
- Easy to adapt progressive needs





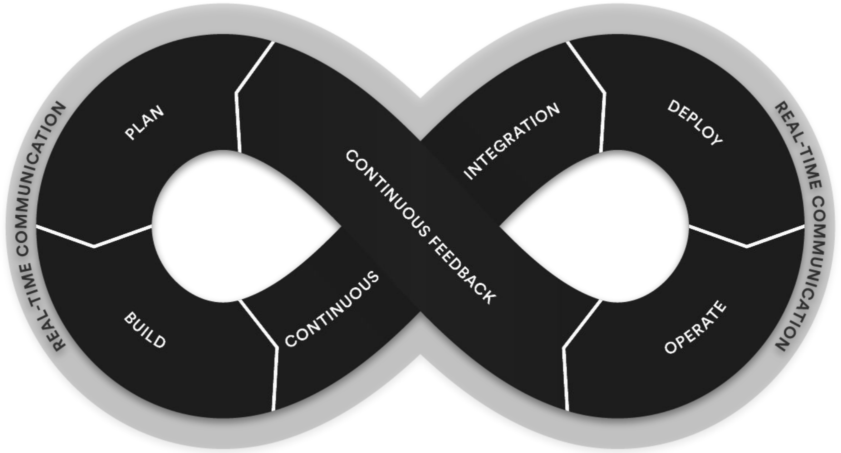
Design Considerations

- Lean
- Segmented
- Redirector(s)
- Interdependence
- Network footprint
- Engagement specific design i.e. C2, domain etc.

DevOps

DevOps Practices

- “DevOps is the combination of cultural philosophies, practices and tools that increases an organization’s ability to deliver applications and services at high velocity”
- DevOps is about creating self-service infrastructure for teams
- Teams that use devops use these practices to automate processes that historically have been manual and slow



Why Automation?

- Design an engagement infra, click around UI, SSH into a server to deploy your designed infrastructure by hand

  For every engagement

Infrastructure-as-code (IAC)

- Write code to define, provision and manage your infrastructure

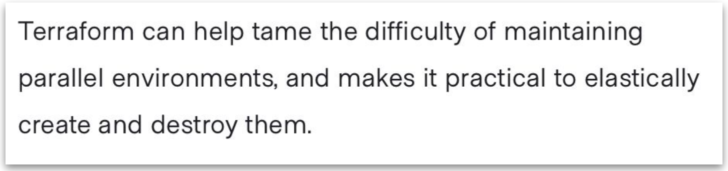
Benefits

- Automate entire provisioning and deployment process
- Represent the state of infrastructure as source files
- Store source files in version control to track changes
- Validate changes
- Remove human element

- Tools
    - Chef
    - Puppet
    - Ansible
    - SaltStack
    - CloudFormation
    - Terraform
- All are open source, backed by large communities and work with many different cloud providers (except AWS CloudFormation)

Why Terraform?

- Chef, Puppet, Ansible and SaltStack are configuration management tools and designed to manage software on existing servers
- CloudFormation and Terraform are provisioning tools and designed to provision the infrastructure
- They can also perform some degree of configuration management
- If coupled with Docker, most of configuration management needs are resolved


Terraform can help tame the difficulty of maintaining parallel environments, and makes it practical to elastically create and destroy them.

- Over time, as you use configuration management tools to apply updates and changes, it leads to a configuration drift
- Using terraform with Docker to deploy changes reduces the likelihood of configuration drift
- Every change is a new deployment

- Terraform encourage a declarative style of coding where terraform itself is responsible for figuring out how to achieve that state
- For example, if you need 10 instances, simplified scripts to use in ansible and terraform is following:

```
- ec2:
   count: 10
   image: ami-v1
   instance_type: t2.micro
```

```
resource "aws_instance"
"redteamops_example" {
 count         = 10
 ami           = "ami-v1"
 instance_type = "t2.micro"
}
```

- If you need to deploy 1 more instance, the scripts will be:

```
- ec2:
   count: 1
   image: ami-v1
   instance_type: t2.micro
```

```
resource "aws_instance"
"redteamops_example" {
 count         = 11
 ami           = "ami-v1"
 instance_type = "t2.micro"
}
```

Previous ansible script is useless when deploying changes while terraform figures the number of instances needs to be deployed itself

- Some tools requires a master and agents to operate
- Terraform is masterless and agentless as it communicates with cloud providers using APIs

Tool Decision

Three common combinations to use

- Provisioning and configuration management
  - Terraform and ansible
- Provisioning and server templating
  - Terraform and packer
- Provisioning, server templating and orchestration
  - Terraform, docker and kubernetes

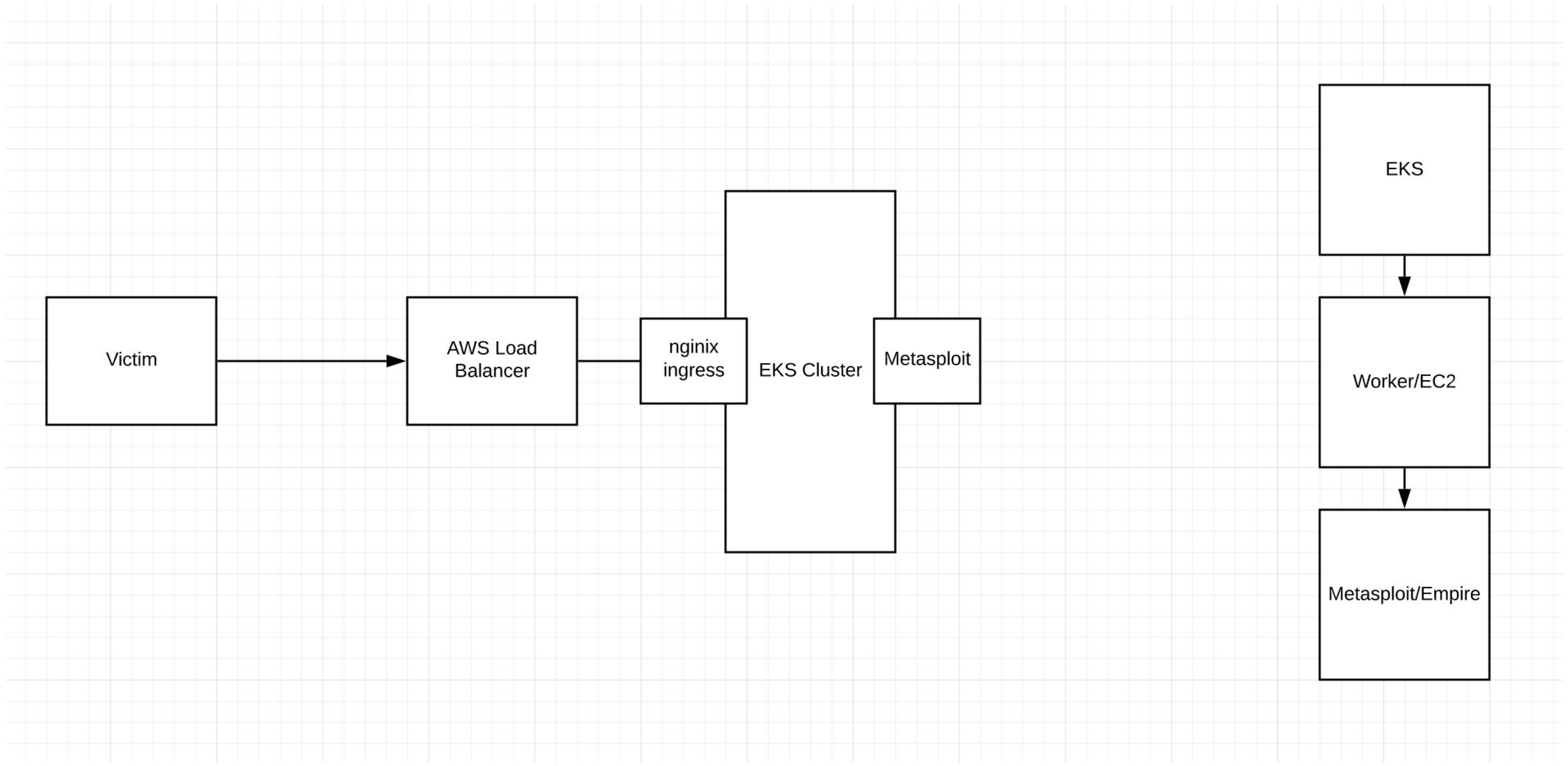
Red Team meets DevOps: Example

github.com/hitcatred/RedTeamOps

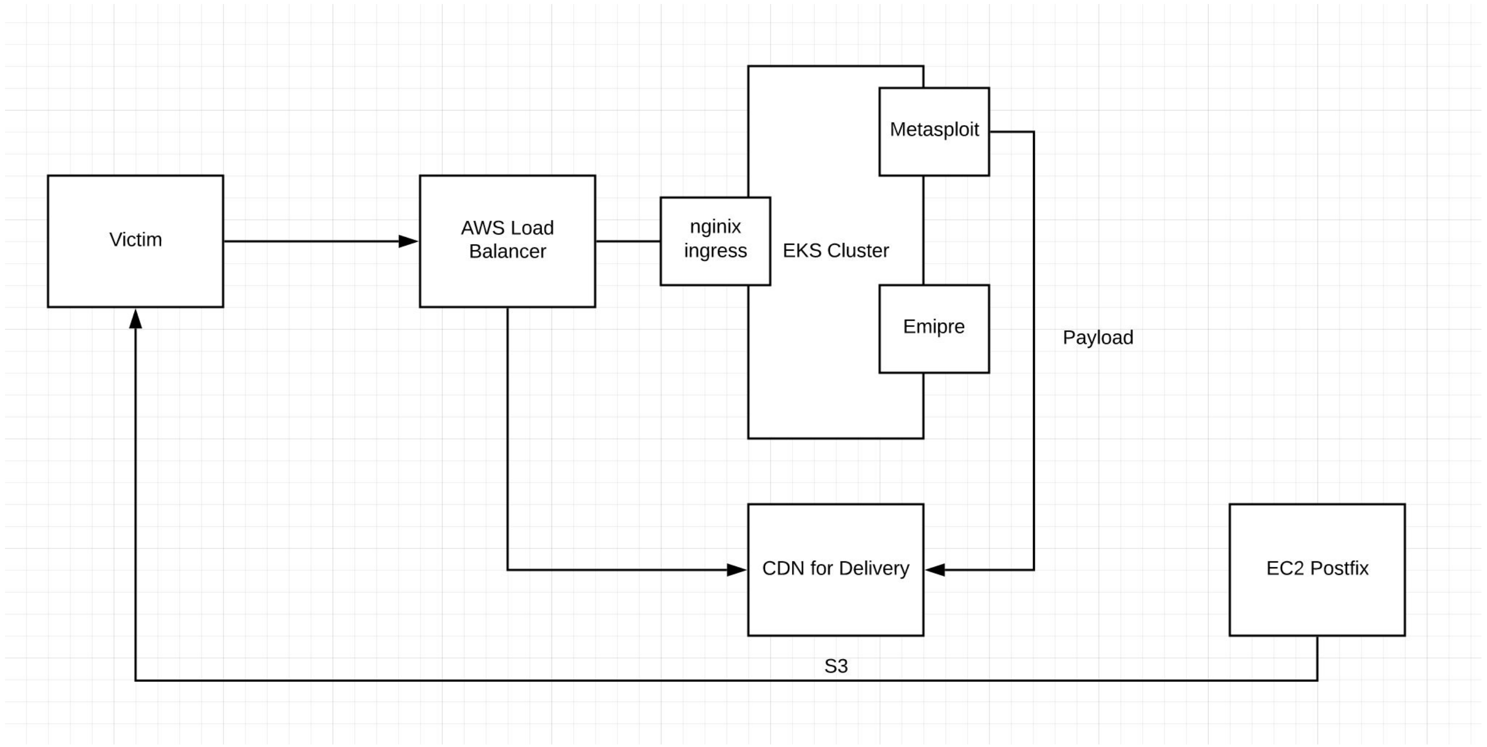
Demo

1. Inspecting infrastructure and scripts
2. Build infrastructure
3. Payload execution
4. Destroy

Simple infrastructure



Extended infrastructure



Maintenance

Script maintenance

- Testing and versioning, changes is a must to avoid surprises

Stages (gitlab-ci.yml)

- Validate
- Deploy
- Destroy

```
terraform_validate:

  image:

        name: hashicorp/terraform:light

        entrypoint:

        - '/usr/bin/env'

        - 'PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin'

stage: test

script:

        - terraform init

        - terraform validate
```

```
terraform_deploy:
  image:
        name: hashicorp/terraform:light
        entrypoint:
        - '/usr/bin/env'
        - 'PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin'
  stage: deploy
  script:
        - terraform init
        - terraform apply --auto-approve
  artifacts:
        paths:
        - terraform.tfstate
        expire_in: 1 day
```

```
terraform_destroy:

  when: always

  image:

            name: hashicorp/terraform:light

            entrypoint:

            - '/usr/bin/env'

            - 'PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin'

stage: destroy

script:

          - terraform init

          - terraform destroy --auto-approve
```

Takeaways

- Choosing correct operation model helps assessing clients’ posture
- Lean, segmented and interdependent infra is the key for a smooth operation
- Use automation and CI to test, document and validate your infra