GET IN THE BOX!

Containerizing Red Team Infrastructure

Dan Astor & Jonn Callahan

BSidesPhilly:2023

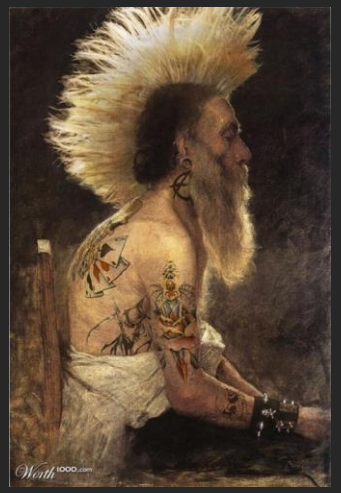
Agenda

Introduction
Common Industry Practices
RedSwarm Overview
Terraform & Ansible & Swarm
Traefik & Routing
Monitoring
Sojourn
Closing Thoughts
Questions

About Us

Jonn Callahan (@atticuss)

- App & cloudsec background
- Enjoys automating things
- Metalhead



Dan Astor (@illegitimateDA)

- Pentesting & Red Team
- Enjoys breaking things
- Not as metal as Jonn

SecurityRisk ADVISORS

Common Industry Practices

Red Team Infrastructure Terminology

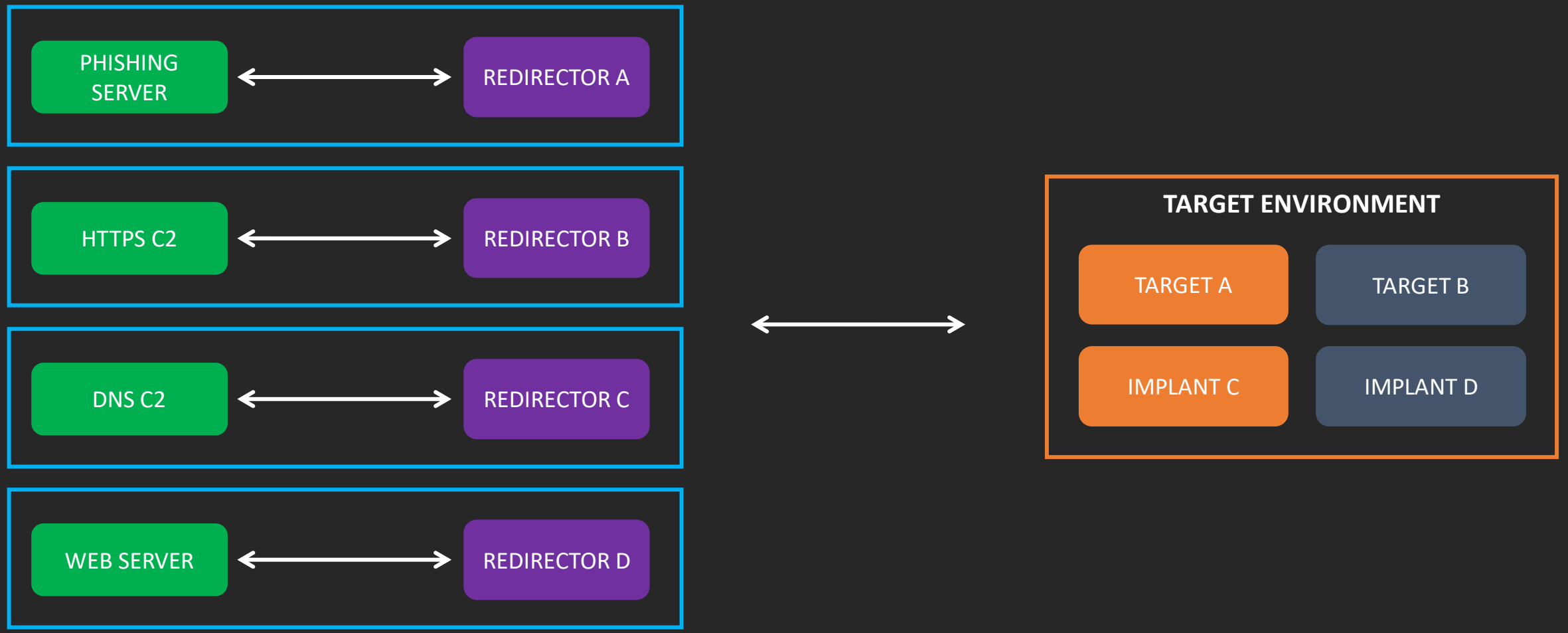
C2 Server: The command and control (C2) system that sends and receives tasks, files, and data from implants.

C2 Channel: The communication channel or protocol used between implant and C2 server (e.g., HTTPS, DNS).

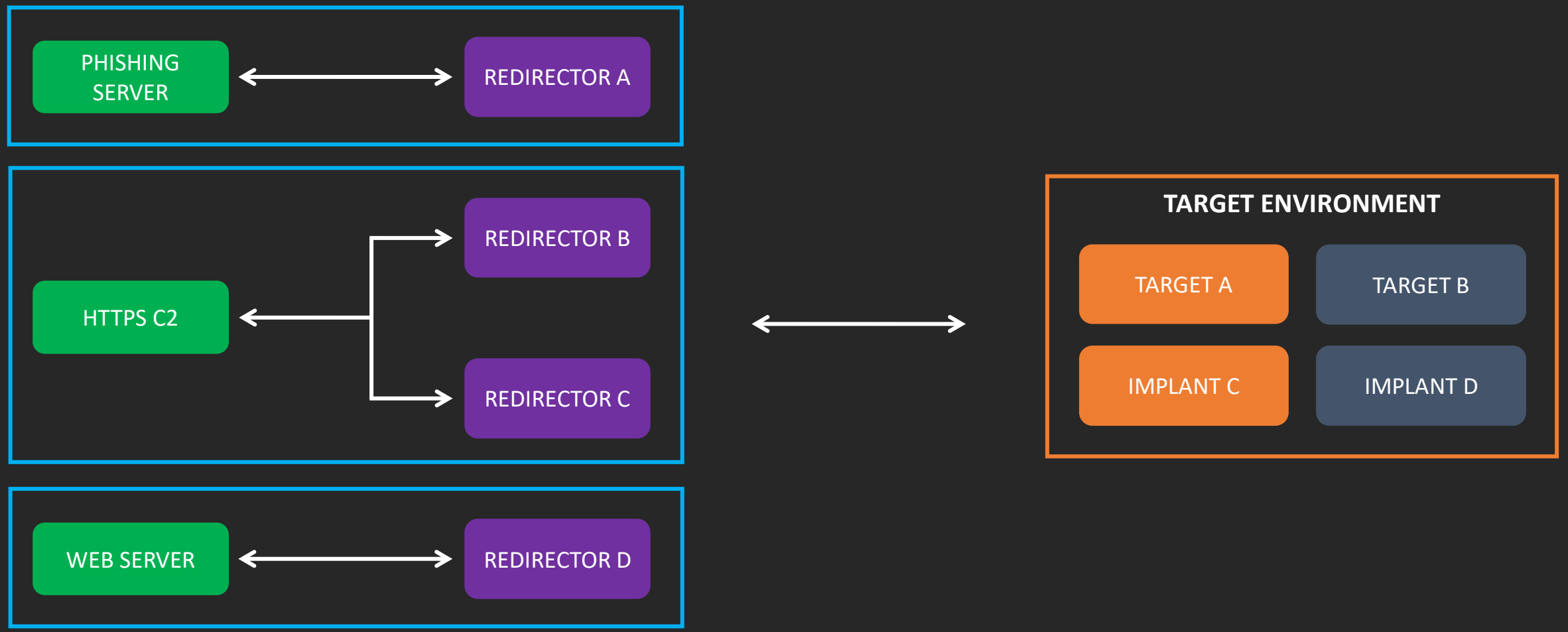
Implant: Piece of code that is created to run on a victim host that connects back to a C2 server.

Redirector: Protects your infrastructure by filtering and routing traffic.

Example Deployment



Example Deployment



Previous Deployment Strategy

Create individual systems for each service (e.g., Redirector, C2, etc.)

1. Define Terraform plan
2. Deploy instances with Terraform
3. Configure instances with Ansible
4. Perform engagement
5. Destroy instances with Terraform

Frustrations & Pitfalls

Higher costs with many single use systems

Maintenance of multiple roles/providers

Ansible OS level configurations

Long build times

Architecture inflexibility

Certificate & domain management

Notable Work

https://github.com/bluscreenofjeff/Red-Team-Infrastructure-Wiki

https://rastamouse.me/infrastructure-as-code-terraform-ansible/

https://github.com/Coalfire-Research/Red-Baron

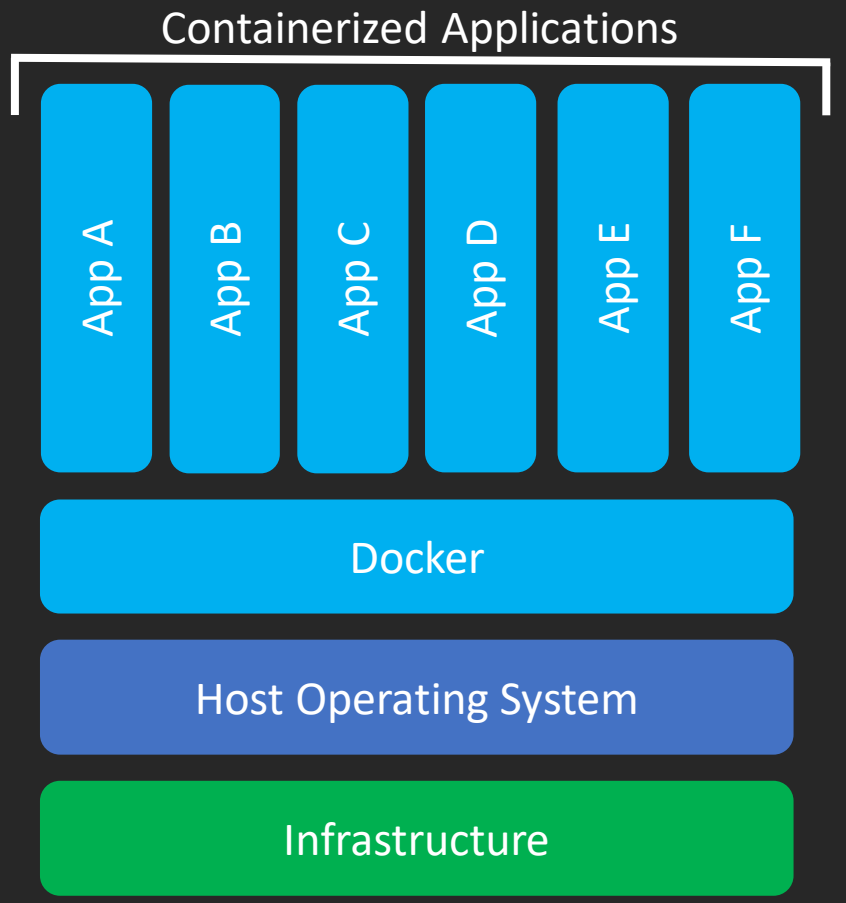
https://www.mdsec.co.uk/2020/02/testing-your-redteam-infrastructure/

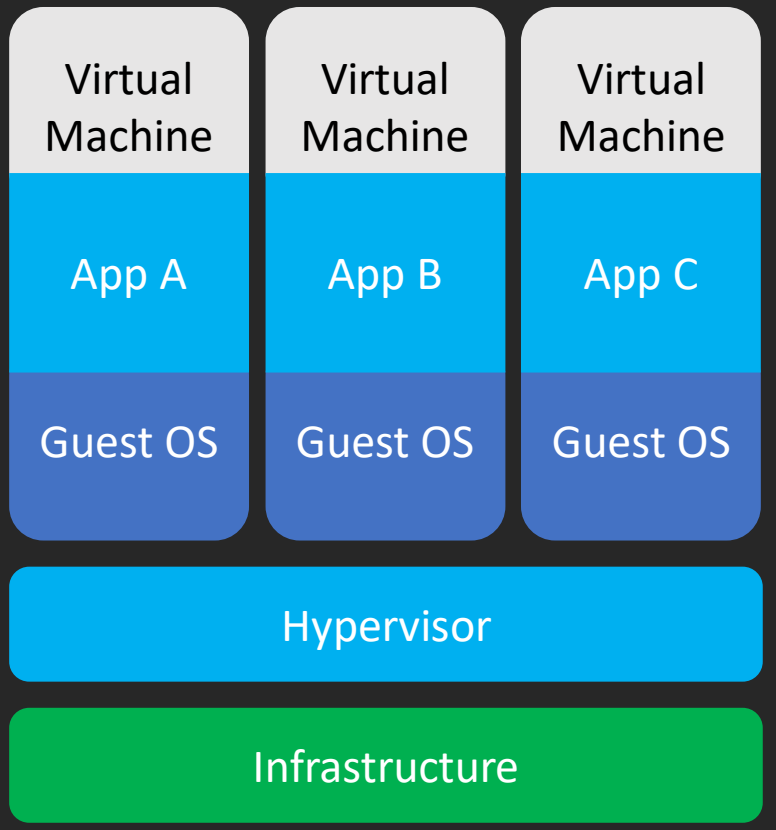
https://github.com/warhorse/warhorse

https://byt3bl33d3r.substack.com/p/taking-the-pain-out-of-c2-infrastructure-3c4

RedSwarm Overview

Containers


Containerized Applications
App A
App B
App C
App D
App E
App F
Docker
Host Operating System
Infrastructure


Virtual Machine
Virtual Machine
Virtual Machine
App A
App B
App C
Guest OS
Guest OS
Guest OS
Hypervisor
Infrastructure

~~Ditch~~ *Limit* Ansible; Embrace Containers

Infrastructure

- (Virtual) hardware layer
- Static: rarely changes or updates after initial design
- Fixed: resources provided are difficult to change

Deploy via Terraform

Configure via Ansible

Services

- Application layer
- Dynamic: constantly being changed and updated
- Scaling: resource requirements fluctuate

Package as Docker containers

Configure via Docker Compose manifests

Overview

Terraform: **Manage** and **provision infrastructure** through code and configuration files vs manual processes

Ansible: Initialize infrastructure via **configuration management**

Docker Images: Package services into isolated **containers**

Docker Swarm Mode: Deploy and manage services via **container orchestration**

Terraform & Ansible & Swarm

Primary Goal: Ease-of-Use

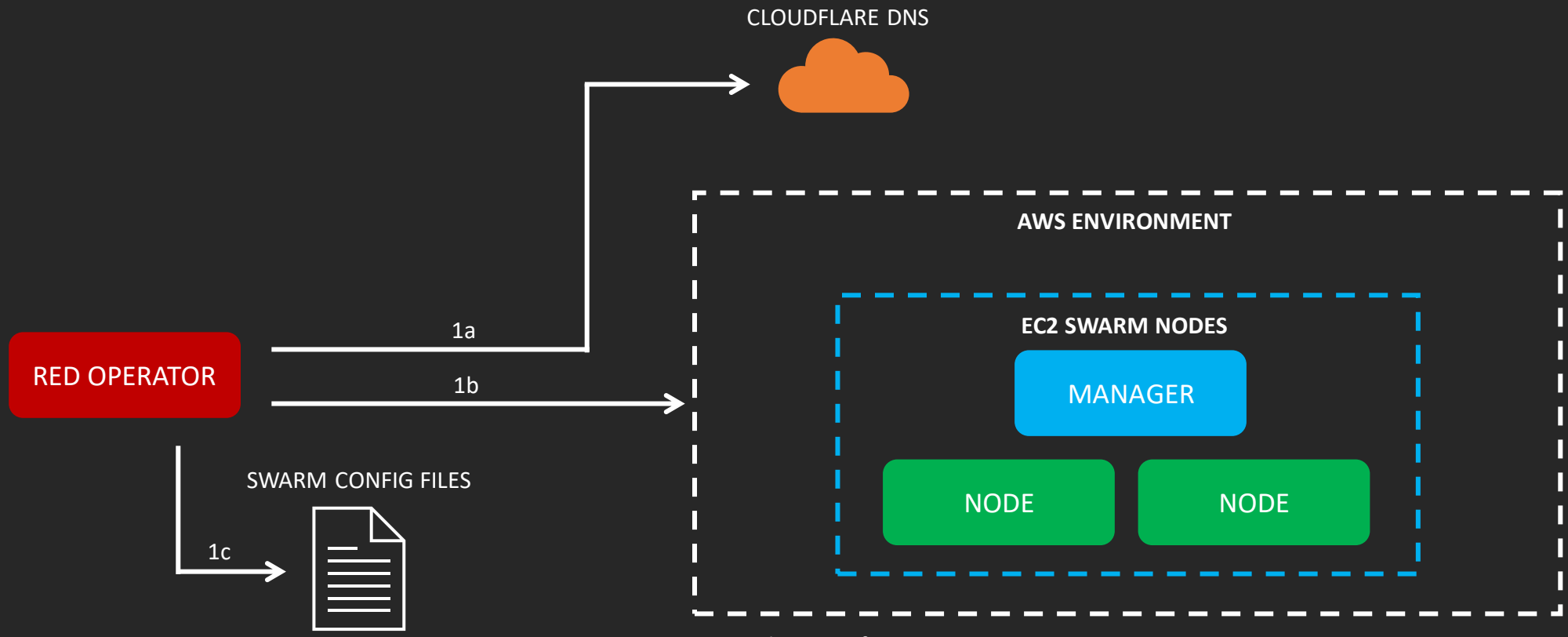
Deployment should be as turnkey as possible:

1. Set all the appropriate values in “terraform.tfvars”
2. `terraform apply -auto-approve`
3. `ansible-playbook –i hosts.ini site.yml`
4. `docker -H ssh://admin@10.10.10.10 stack deploy -c traefik.yaml traefik`

Repeat the “stack deploy” for each desired service

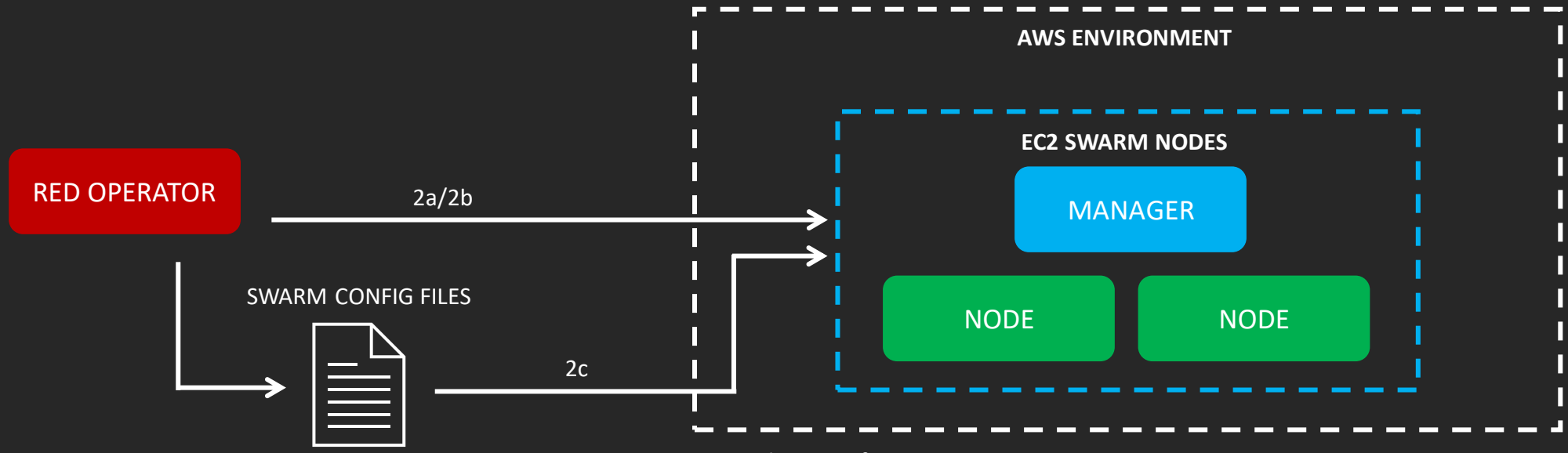
Deployment Workflow

1. Run `terraform apply`
    a) Creates DNS records
    b) Creates EC2 instances
    c) Creates swarm config files



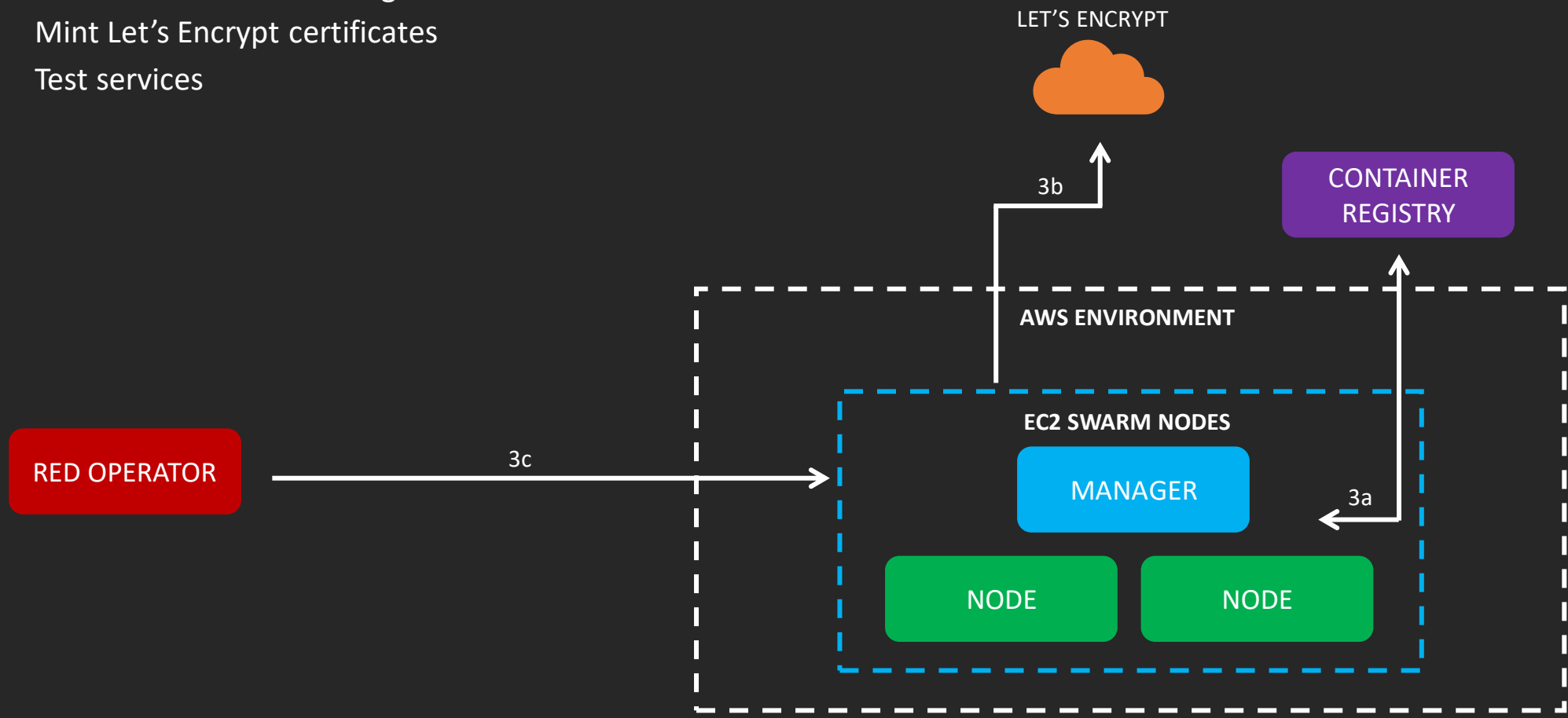
Deployment Workflow

2. Run `ansible-playbook`
    a) Configure each node
    b) Initialize the swarm
    c) Deploy Swarm services



Deployment Workflow

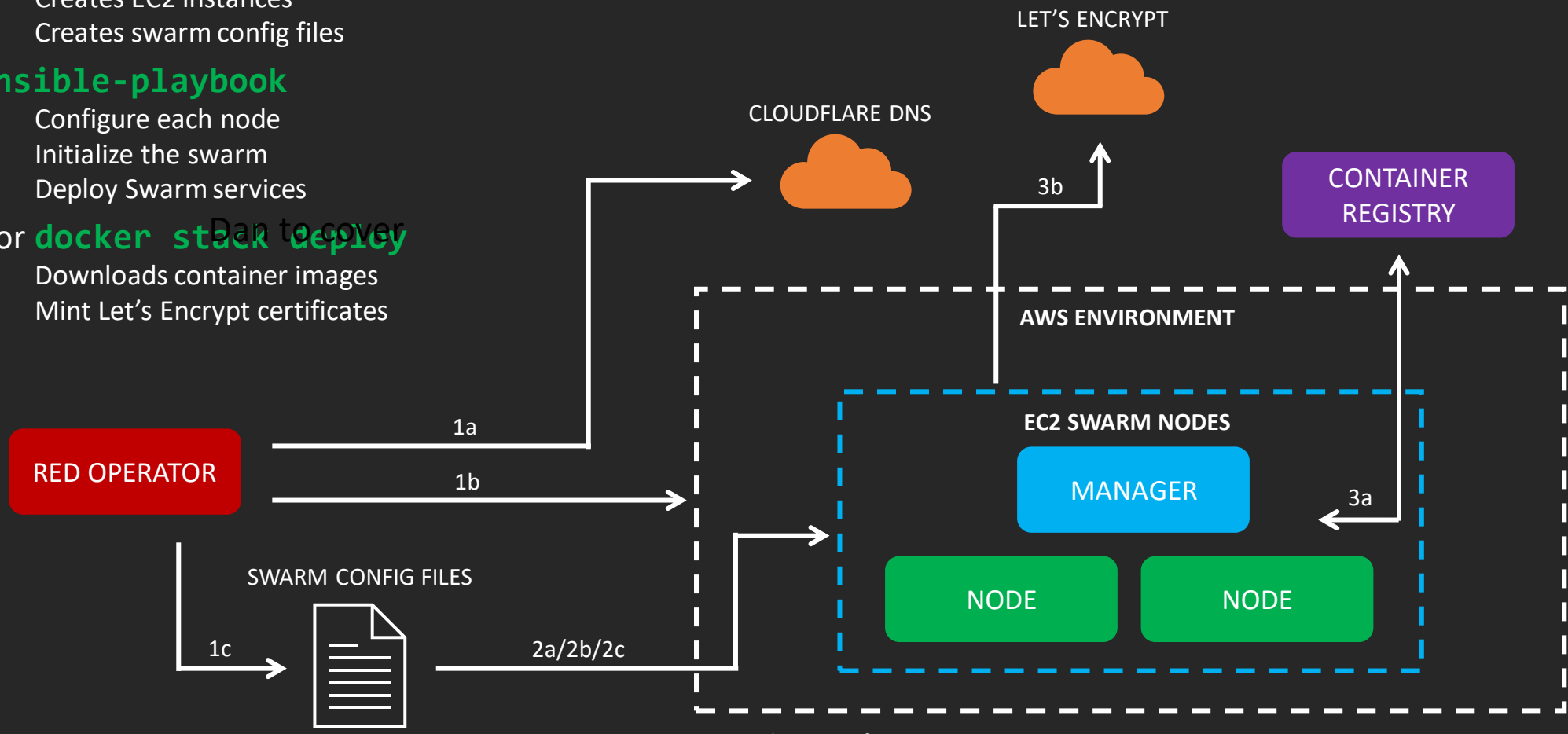
3. Wait for `docker stack deploy`
    a) Downloads container images
    b) Mint Let's Encrypt certificates
    c) Test services



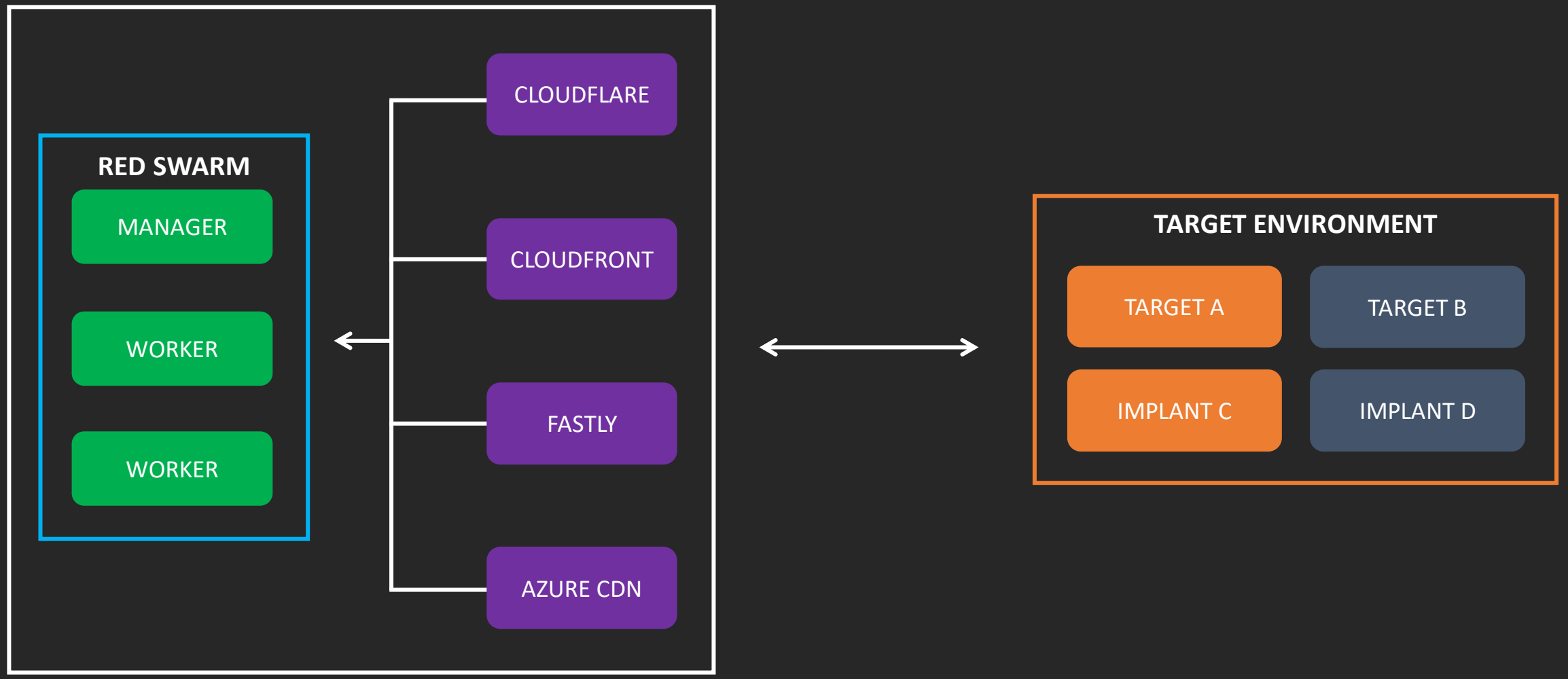
Deployment Workflow

1. Run `terraform apply`
   a) Creates DNS records
   b) Creates EC2 instances
   c) Creates swarm config files
2. Run `ansible-playbook`
   a) Configure each node
   b) Initialize the swarm
   c) Deploy Swarm services
3. Wait for `docker stack deploy`
   a) Downloads container images
   b) Mint Let's Encrypt certificates

Dan to cover



Example Deployment



Docker Images

Core services (e.g., C2 server)

- Internally maintained
- Housed within private ECR instances
- Regular & automatic pipeline builds

Metaservices (e.g., Traefik)

- Maintained by “trusted” third-party
- Housed within Docker Hub
- Version pinned

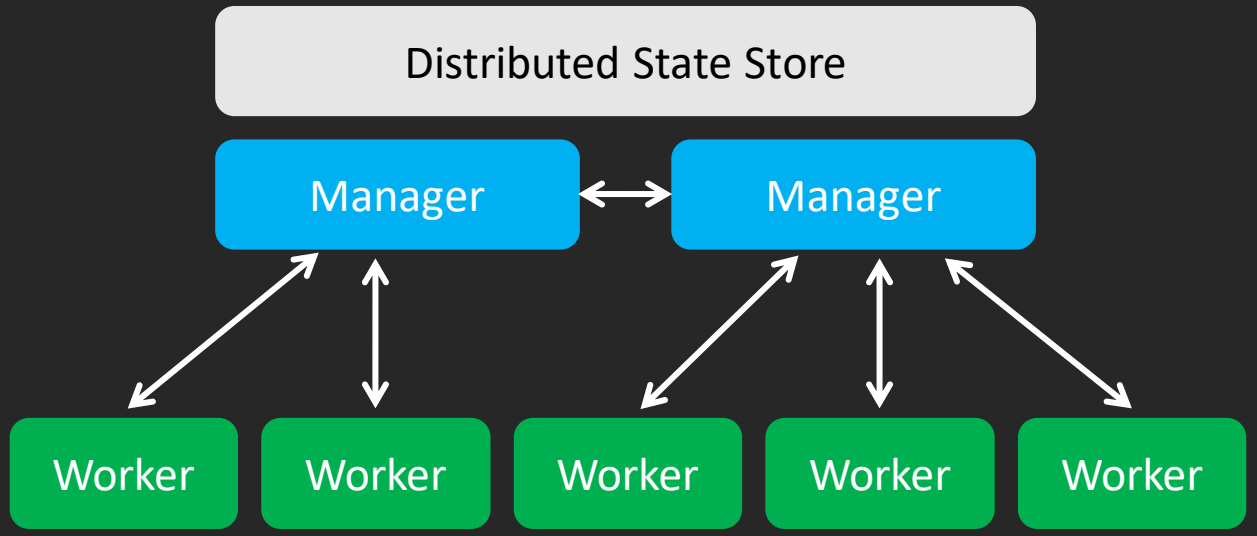
```
# Dockerfile
FROM golang:1.18.6
WORKDIR /opt
RUN git clone https://github.com/kgretzky/evilginx2.git && \
        cd evilginx2 && \
        make
COPY ./start.sh /opt
CMD [ "/opt/start.sh" ]


# start.sh
#!/bin/bash
screen -d -m /opt/evilginx2/bin/evilginx -p /opt/evilginx2/phishlets/
```

Docker Swarm

Container orchestration engine

Decouples applications from underlying servers

Kubernetes is overkill



Stacks

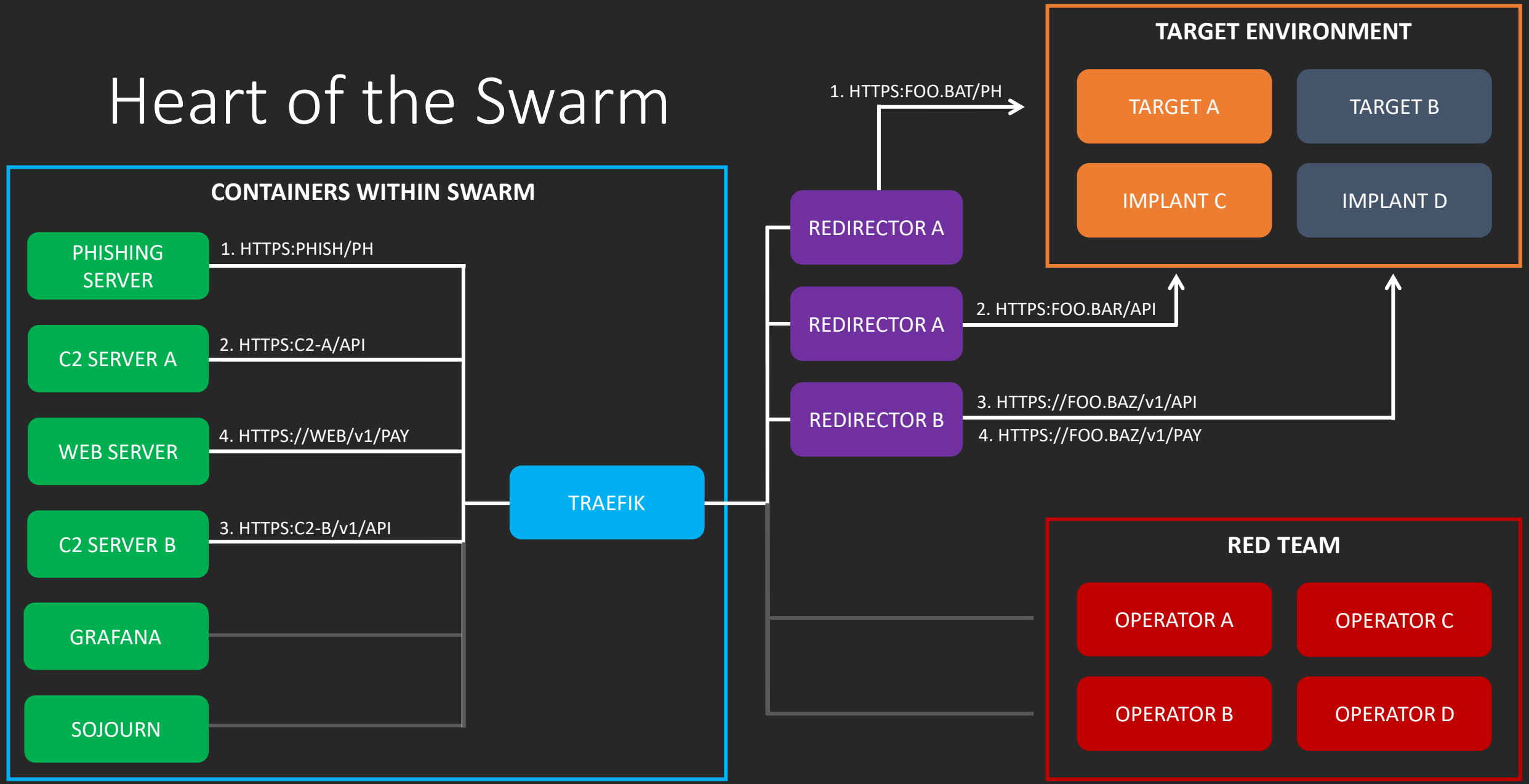
Groups images & network configurations together into a set of services

Consumes Docker Compose v3 files

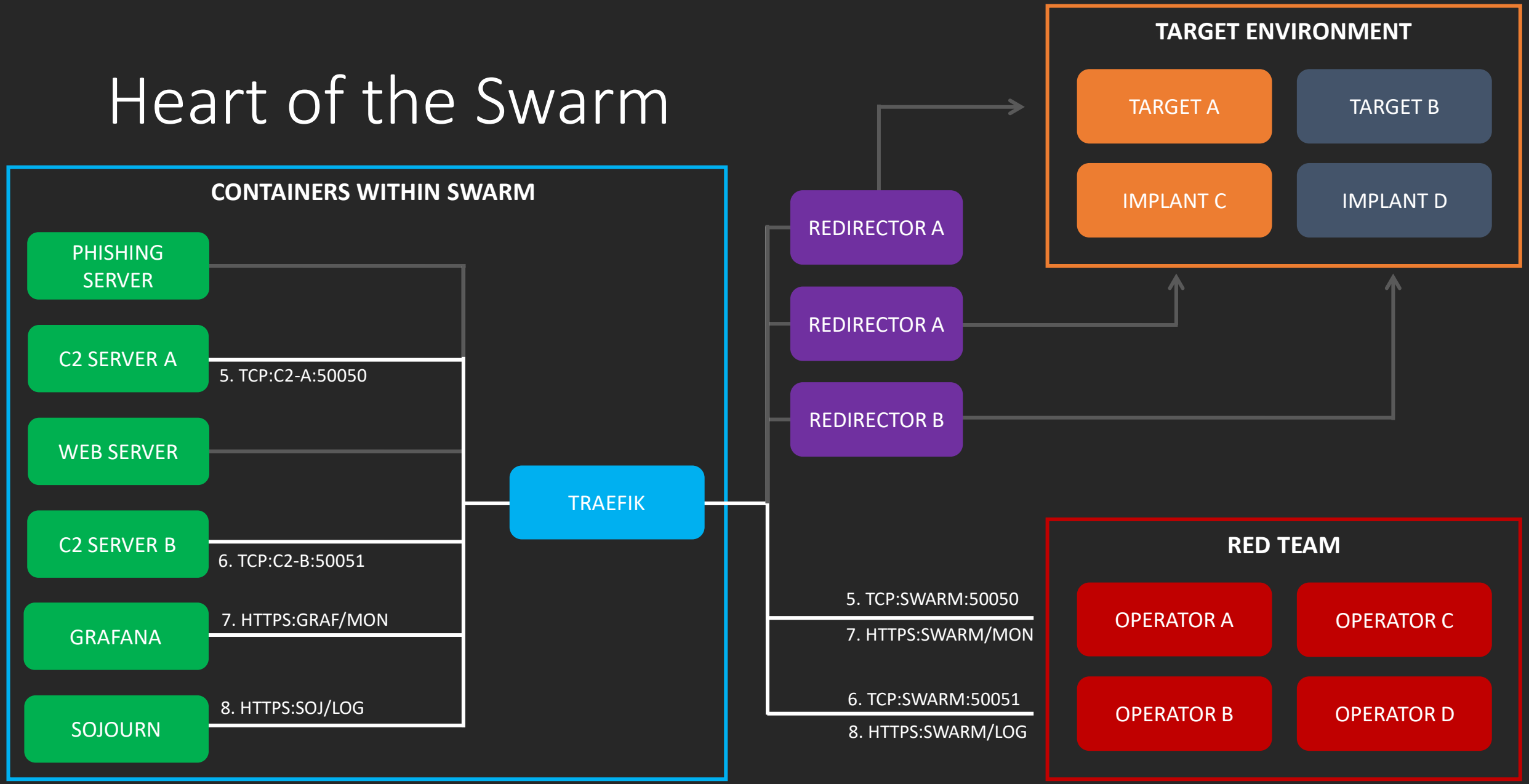
Example deploy command:

```
docker -H ssh://admin@swarm.host stack deploy –c
portainer.yaml portainer --with-registry-auth
```

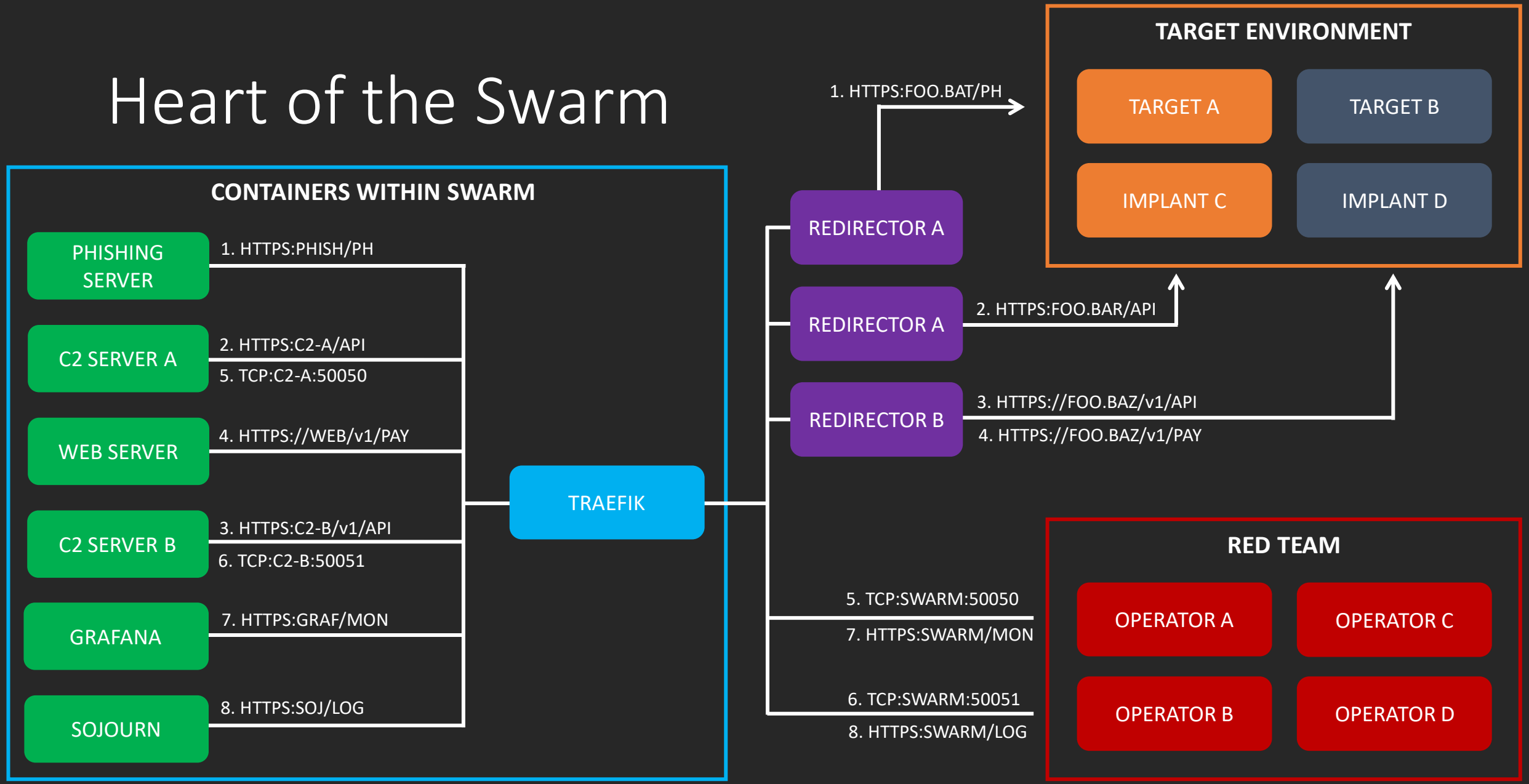
Heart of the Swarm



Heart of the Swarm

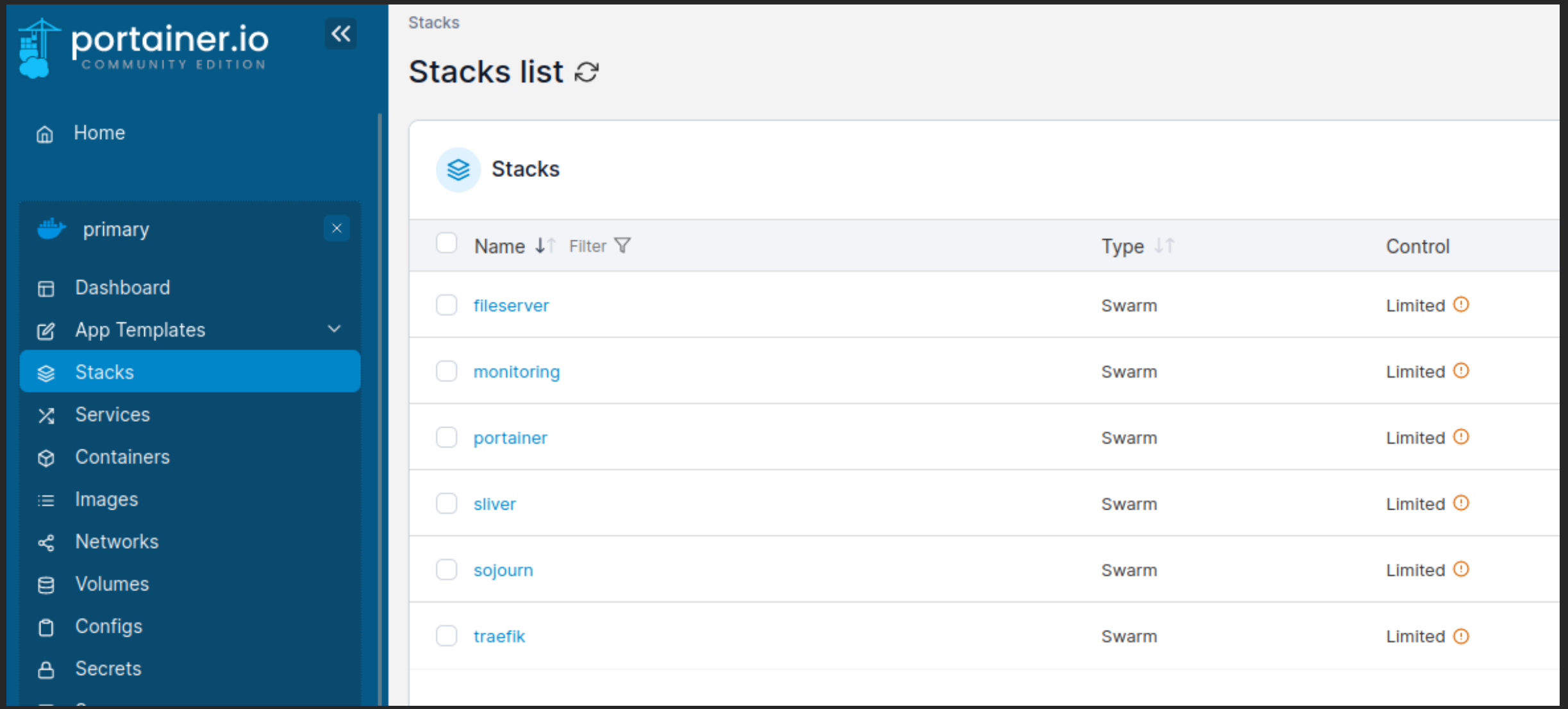


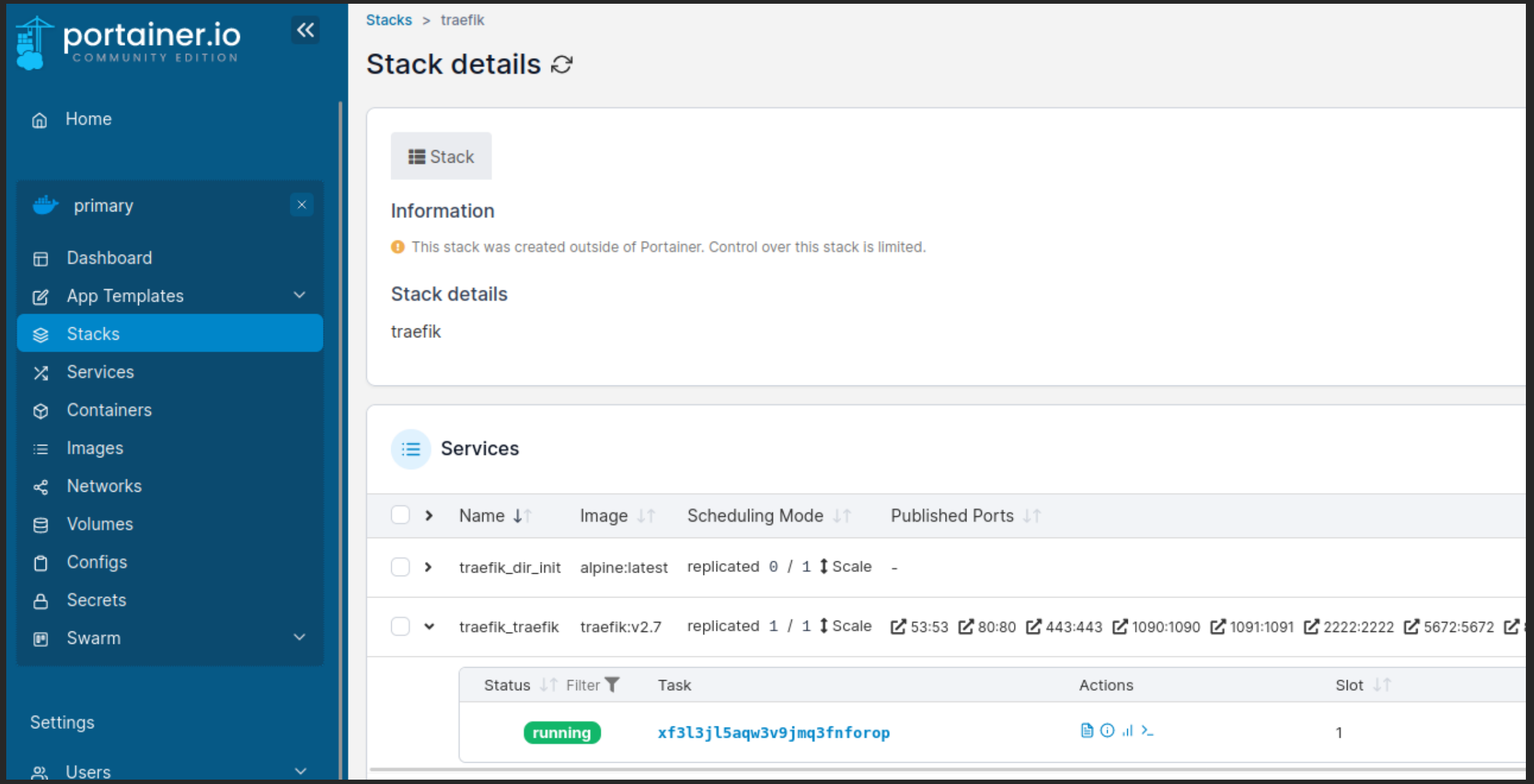
Heart of the Swarm

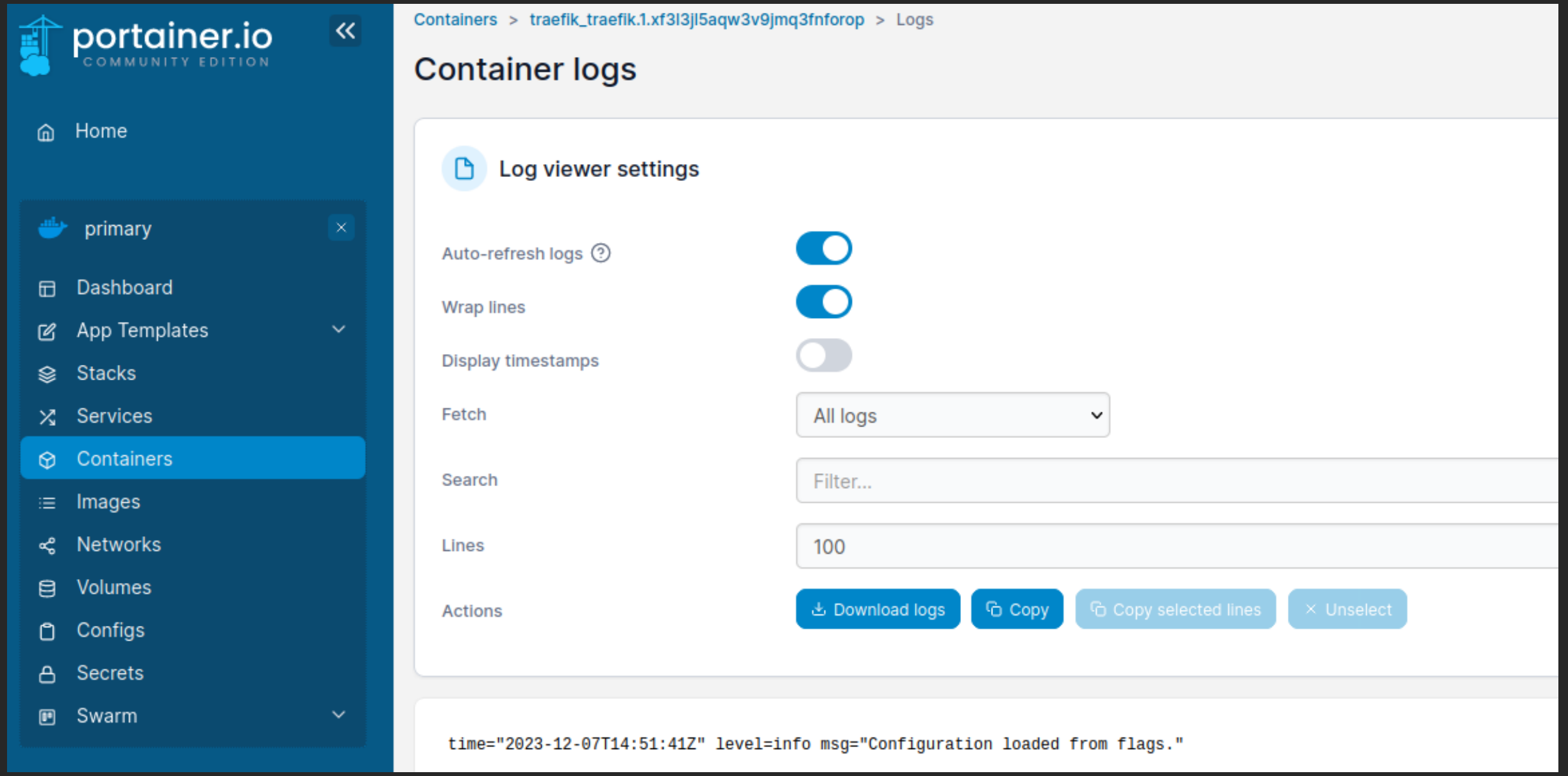


Portainer

- Container used for easy management of the Swarm
- Helpful for deployment, management, debugging


portainer.io
COMMUNITY EDITION
Home
primary
Dashboard
App Templates
Stacks
Services
Containers
Images
Networks
Volumes
Configs
Secrets
Stacks
Stacks list
Stacks
Name
Filter
Type
Control
fileserver
Swarm
Limited
monitoring
Swarm
Limited
portainer
Swarm
Limited
sliver
Swarm
Limited
sojourn
Swarm
Limited
traefik
Swarm
Limited


portainer.io
COMMUNITY EDITION
Home
primary
Dashboard
App Templates
Stacks
Services
Containers
Images
Networks
Volumes
Configs
Secrets
Swarm
Settings
Users
Stacks > traefik
Stack details
Stack
Information
This stack was created outside of Portainer. Control over this stack is limited.
Stack details
traefik
Services
Name
Image
Scheduling Mode
Published Ports
traefik_dir_init
alpine:latest
replicated 0 / 1 Scale
-
traefik_traefik
traefik:v2.7
replicated 1 / 1 Scale
53:53
80:80
443:443
1090:1090
1091:1091
2222:2222
5672:5672
Status
Filter
Task
Actions
Slot
running
xf3l3jl5aqw3v9jmq3fnforop
1


portainer.io
COMMUNITY EDITION
Home
primary
Dashboard
App Templates
Stacks
Services
Containers
Images
Networks
Volumes
Configs
Secrets
Swarm
Containers > traefik_traefik.1.xf3l3jl5aqw3v9jmq3fnforop > Logs
Container logs
Log viewer settings
Auto-refresh logs
Wrap lines
Display timestamps
Fetch
All logs
Search
Filter...
Lines
100
Actions
Download logs
Copy
Copy selected lines
Unselect
time="2023-12-07T14:51:41Z" level=info msg="Configuration loaded from flags."

Traefik and Routing


C2 traffic
Phishing traffic
Canary traffic
Traefik

Traefik

- Container used for HTTP/TCP/UDP routing
- Uses ports, rules, services, and routes to direct traffic
- Configure routes via container labels
- Auto-discover Swarm services
- Automatically mints & renews Let’s Encrypt certificates

```
labels:
  - traefik.enable=true
  - traefik.docker.network=public
  - traefik.constraint-label=public
  # primary profile
  - traefik.http.routers.sliver-profile.rule=Host(${sliver_profile_domains}) &&
      PathPrefix(${sliver_profile_paths}) && Headers(`User-Agent`,
      `${sliver_profile_ua}`)
  - traefik.http.routers.sliver-profile.entrypoints=https
  - traefik.http.routers.sliver-profile.tls=true
  - traefik.http.routers.sliver-profile.tls.certresolver=le
  - traefik.http.routers.sliver-profile.service=sliver-https
  - traefik.http.services.sliver-https.loadbalancer.server.port=443
  - traefik.http.services.sliver-https.loadbalancer.server.scheme=https
  # multiplayer connect
  - traefik.tcp.routers.sliver-multiplayer.rule=HostSNI(`*`)
  - traefik.tcp.routers.sliver-multiplayer.entrypoints=operator-ingress-tcp
  - traefik.tcp.routers.sliver-multiplayer=sliver-multiplayer
  - traefik.tcp.services.sliver-multiplayer.server.port=31337
```


```
labels:
  - traefik.enable=true
  - traefik.docker.network=public
  - traefik.constraint-label=public
  # primary profile
  - traefik.http.routers.sliver-profile.rule=Host(${sliver_profile_domains}) &&
      PathPrefix(${sliver_profile_paths}) && Headers(`User-Agent`,
      `${sliver_profile_ua}`)
  - traefik.http.routers.sliver-profile.entrypoints=https
  - traefik.http.routers.sliver-profile.tls=true
  - traefik.http.routers.sliver-profile.tls.certresolver=le
  - traefik.http.routers.sliver-profile.service=sliver-https
  - traefik.http.services.sliver-https.loadbalancer.server.port=443
  - traefik.http.services.sliver-https.loadbalancer.server.scheme=https
  # multiplayer connect
  - traefik.tcp.routers.sliver-multiplayer.rule=HostSNI(`*`)
  - traefik.tcp.routers.sliver-multiplayer.entrypoints=operator-ingress-tcp
  - traefik.tcp.routers.sliver-multiplayer=sliver-multiplayer
  - traefik.tcp.services.sliver-multiplayer.server.port=31337
```

```
labels:
  - traefik.enable=true
  - traefik.docker.network=public
  - traefik.constraint-label=public
  # primary profile
  - traefik.http.routers.sliver-profile.rule=Host(${sliver_profile_domains}) &&
      PathPrefix(${sliver_profile_paths}) && Headers(`User-Agent`,
      `${sliver_profile_ua}`)
  - traefik.http.routers.sliver-profile.entrypoints=https
  - traefik.http.routers.sliver-profile.tls=true
  - traefik.http.routers.sliver-profile.tls.certresolver=le
  - traefik.http.routers.sliver-profile.service=sliver-https
  - traefik.http.services.sliver-https.loadbalancer.server.port=443
  - traefik.http.services.sliver-https.loadbalancer.server.scheme=https
  # multiplayer connect
  - traefik.tcp.routers.sliver-multiplayer.rule=HostSNI(`*`)
  - traefik.tcp.routers.sliver-multiplayer.entrypoints=operator-ingress-tcp
  - traefik.tcp.routers.sliver-multiplayer=sliver-multiplayer
  - traefik.tcp.services.sliver-multiplayer.server.port=31337
```

2

```
labels:
  - traefik.enable=true
  - traefik.docker.network=public
  - traefik.constraint-label=public
  # primary profile
  - traefik.http.routers.sliver-profile.rule=Host(${sliver_profile_domains}) &&
      PathPrefix(${sliver_profile_paths}) && Headers(`User-Agent`,
      `${sliver_profile_ua}`)
  - traefik.http.routers.sliver-profile.entrypoints=https
  - traefik.http.routers.sliver-profile.tls=true
  - traefik.http.routers.sliver-profile.tls.certresolver=le
  - traefik.http.routers.sliver-profile.service=sliver-https
  - traefik.http.services.sliver-https.loadbalancer.server.port=443
  - traefik.http.services.sliver-https.loadbalancer.server.scheme=https
  # multiplayer connect
  - traefik.tcp.routers.sliver-multiplayer.rule=HostSNI(`*`)
  - traefik.tcp.routers.sliver-multiplayer.entrypoints=operator-ingress-tcp
  - traefik.tcp.routers.sliver-multiplayer=sliver-multiplayer
  - traefik.tcp.services.sliver-multiplayer.server.port=31337
```

3

```
labels:
  - traefik.enable=true
  - traefik.docker.network=public
  - traefik.constraint-label=public
  # primary profile
  - traefik.http.routers.sliver-profile.rule=Host(${sliver_profile_domains}) &&
      PathPrefix(${sliver_profile_paths}) && Headers(`User-Agent`,
      `${sliver_profile_ua}`)
  - traefik.http.routers.sliver-profile.entrypoints=https
  - traefik.http.routers.sliver-profile.tls=true
  - traefik.http.routers.sliver-profile.tls.certresolver=le
  - traefik.http.routers.sliver-profile.service=sliver-https
  - traefik.http.services.sliver-https.loadbalancer.server.port=443
  - traefik.http.services.sliver-https.loadbalancer.server.scheme=https
  # multiplayer connect
  - traefik.tcp.routers.sliver-multiplayer.rule=HostSNI(`*`)
  - traefik.tcp.routers.sliver-multiplayer.entrypoints=operator-ingress-tcp
  - traefik.tcp.routers.sliver-multiplayer=sliver-multiplayer
  - traefik.tcp.services.sliver-multiplayer.server.port=31337
```

4

Monitoring

Infrastructure Monitoring

Infrastructure telemetry

- Traefik access logs
- System and service health

Application telemetry

- Phishing logs
- Operator logs
- Canary logs

Logging & Metrics

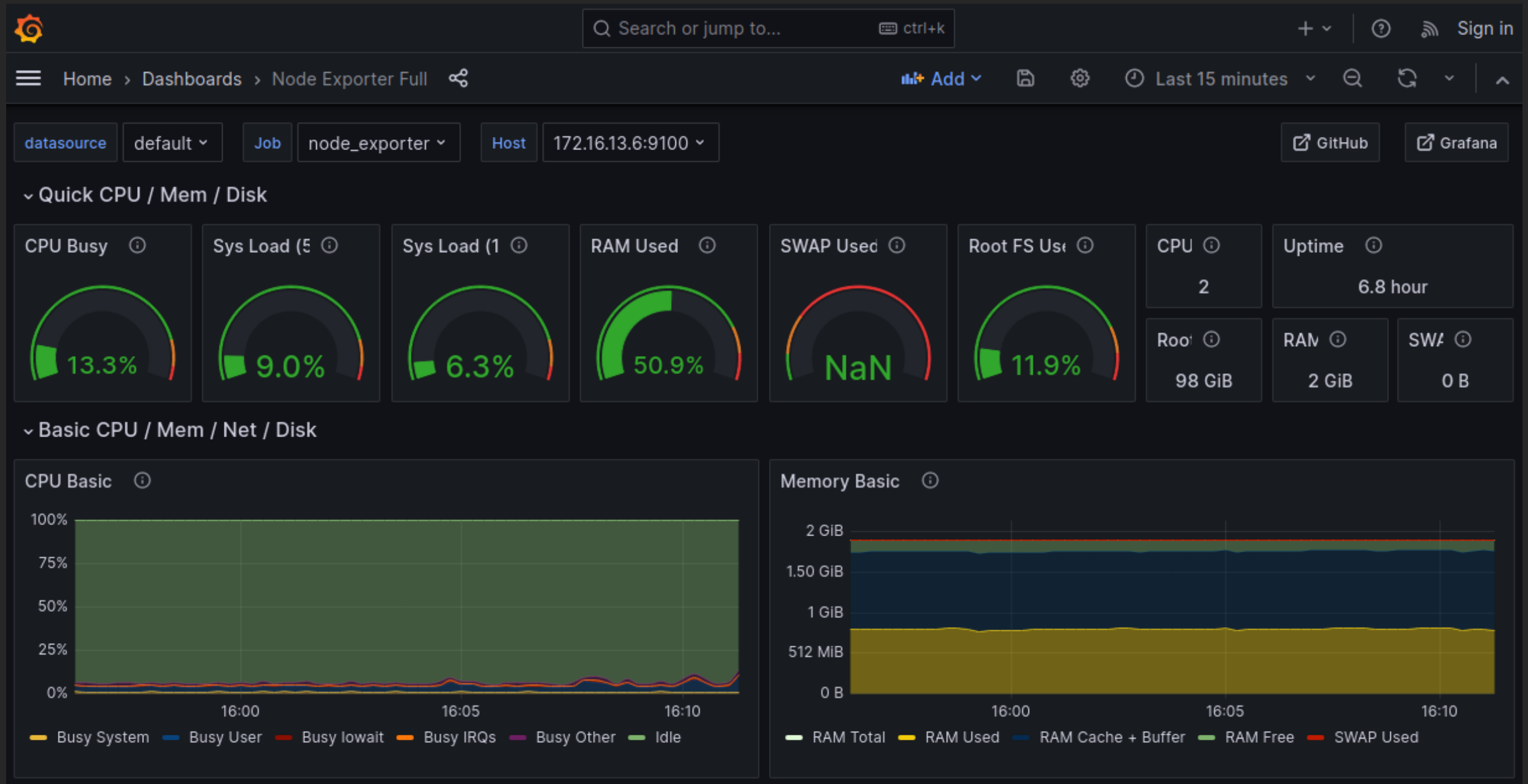
Prometheus + Loki

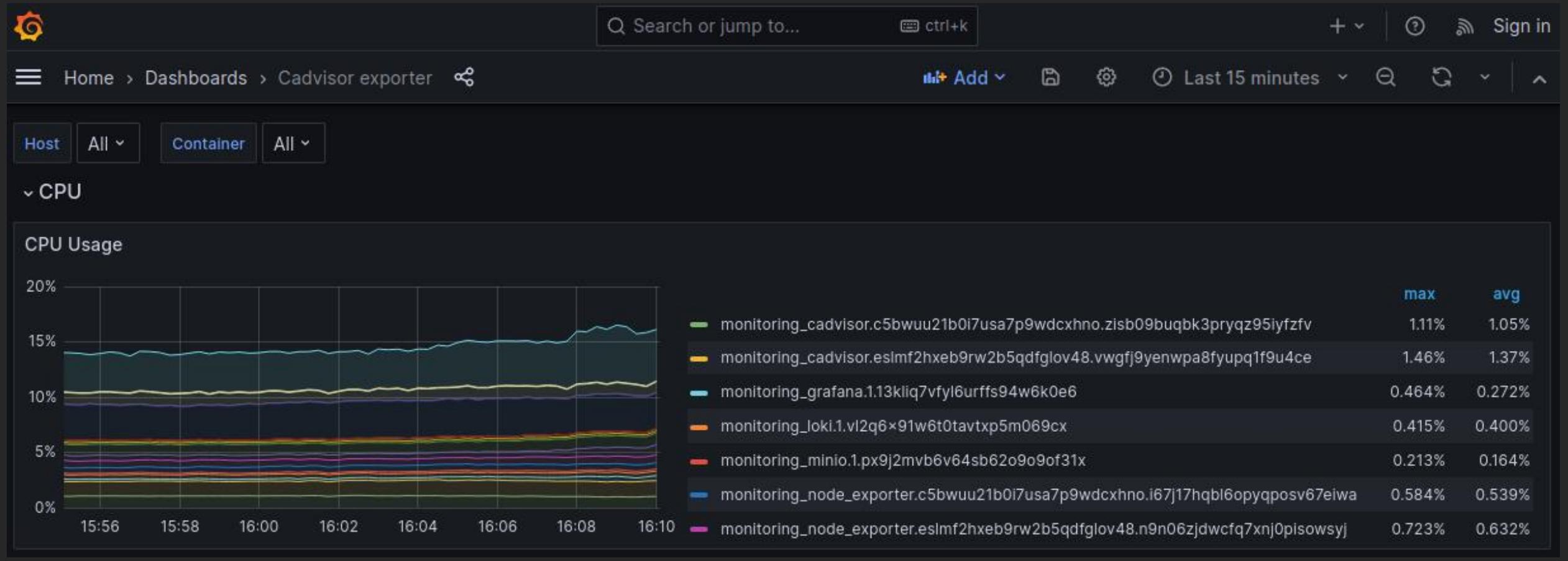
- ✓ Lightweight
- ✓ Easily integrates with our stack
- ✓ Infrastructure setup is turnkey
- ✓ Log ingestion is turnkey
- ✗ Limited pipeline flexibility

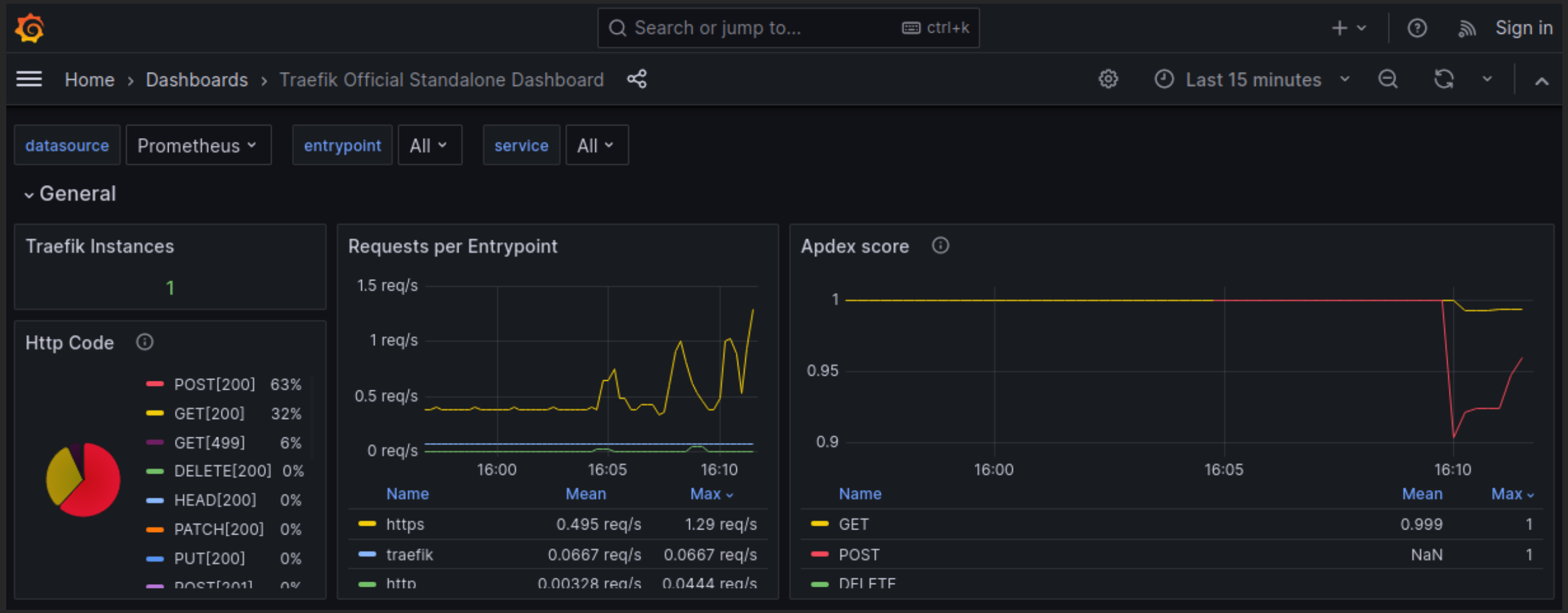
Red Team SIEM || RedELK

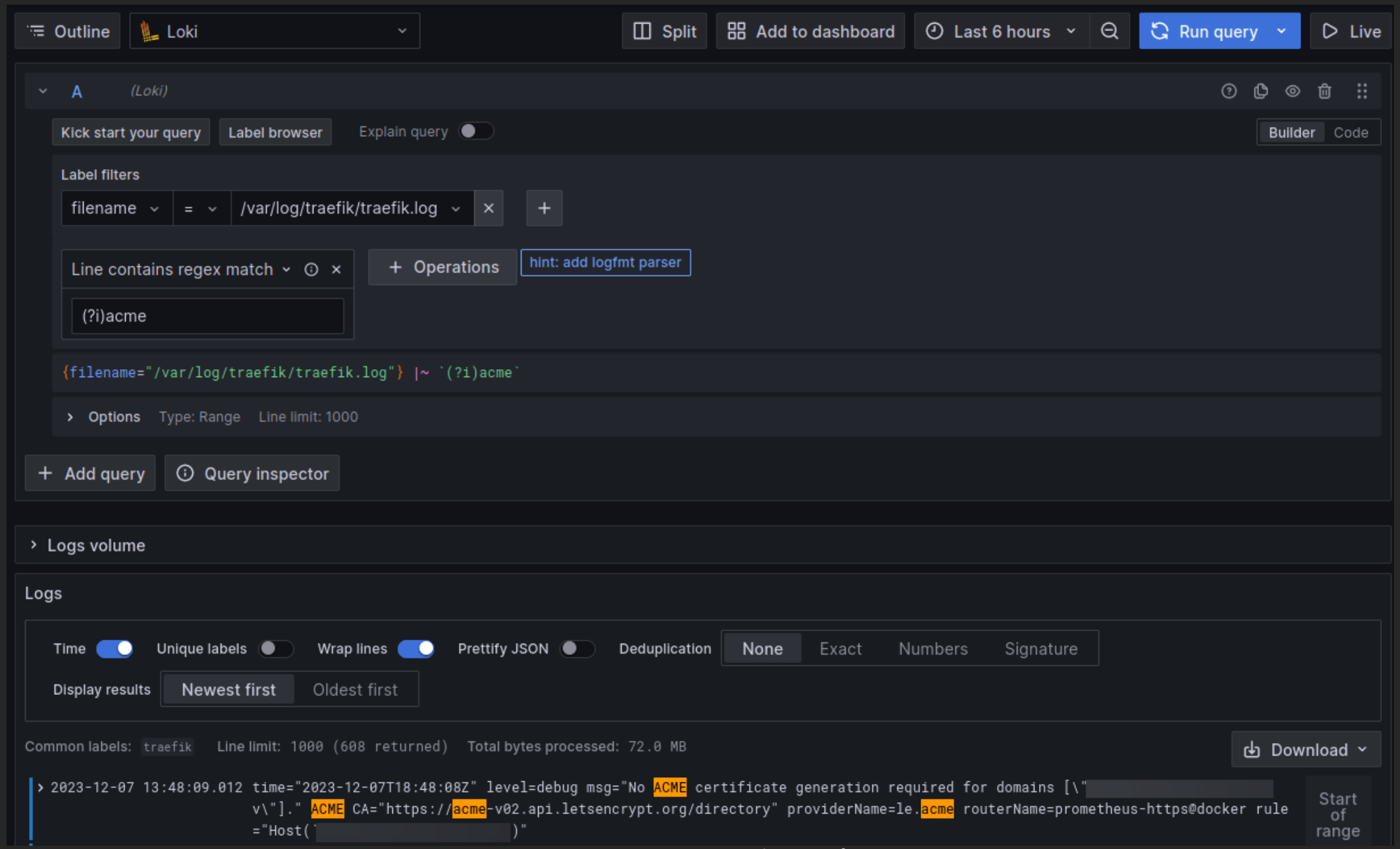
- ✗ ELK is resource intensive
- ✗ Elasticsearch is complex
- ✓ Infrastructure setup is turnkey
- ✗ Log ingestion is not turnkey
- ✓ Log parsing for most red tools

https://github.com/outflanknl/RedELK
https://github.com/SecurityRiskAdvisors/RedTeamSIEM


Search or jump to... ctrl+k
Sign in
Home › Dashboards › Node Exporter Full
Add
Last 15 minutes
datasource default
Job node_exporter
Host 172.16.13.6:9100
GitHub
Grafana
Quick CPU / Mem / Disk
CPU Busy
13.3%
Sys Load (5
9.0%
Sys Load (1
6.3%
RAM Used
50.9%
SWAP Used
NaN
Root FS Use
11.9%
CPU
2
Uptime
6.8 hour
Root
98 GiB
RAM
2 GiB
SWA
0 B
Basic CPU / Mem / Net / Disk
CPU Basic
100%
75%
50%
25%
0%
16:00
16:05
16:10
Busy System
Busy User
Busy Iowait
Busy IRQs
Busy Other
Idle
Memory Basic
2 GiB
1.50 GiB
1 GiB
512 MiB
0 B
16:00
16:05
16:10
RAM Total
RAM Used
RAM Cache + Buffer
RAM Free
SWAP Used


Search or jump to... ctrl+k
Sign in
Home › Dashboards › Cadvisor exporter
Add
Last 15 minutes
Host All
Container All
CPU
CPU Usage
20%
15%
10%
5%
0%
15:56
15:58
16:00
16:02
16:04
16:06
16:08
16:10
max
avg
monitoring_cadvisor.c5bwuu21b0i7usa7p9wdcxhno.zisb09buqbk3pryqz95iyfzfv 1.11% 1.05%
monitoring_cadvisor.eslmf2hxeb9rw2b5qdfglov48.vwgfj9yenwpa8fyupq1f9u4ce 1.46% 1.37%
monitoring_grafana.1.13kliq7vfyl6urffs94w6k0e6 0.464% 0.272%
monitoring_loki.1.vl2q6×91w6t0tavtxp5m069cx 0.415% 0.400%
monitoring_minio.1.px9j2mvb6v64sb62o9o9of31x 0.213% 0.164%
monitoring_node_exporter.c5bwuu21b0i7usa7p9wdcxhno.i67j17hqbl6opyqposv67eiwa 0.584% 0.539%
monitoring_node_exporter.eslmf2hxeb9rw2b5qdfglov48.n9n06zjdwcfq7xnj0pisowsyj 0.723% 0.632%


Search or jump to... ctrl+k
Sign in
Home › Dashboards › Traefik Official Standalone Dashboard
Last 15 minutes
datasource Prometheus
entrypoint All
service All
General
Traefik Instances
1
Http Code
POST[200] 63%
GET[200] 32%
GET[499] 6%
DELETE[200] 0%
HEAD[200] 0%
PATCH[200] 0%
PUT[200] 0%
Requests per Entrypoint
1.5 req/s
1 req/s
0.5 req/s
0 req/s
16:00
16:05
16:10
Name Mean Max
https 0.495 req/s 1.29 req/s
traefik 0.0667 req/s 0.0667 req/s
Apdex score
1
0.95
0.9
16:00
16:05
16:10
Name Mean Max
GET 0.999 1
POST NaN 1


Outline
Loki
Split
Add to dashboard
Last 6 hours
Run query
Live
A (Loki)
Kick start your query
Label browser
Explain query
Builder
Code
Label filters
filename
=
/var/log/traefik/traefik.log
Line contains regex match
(?i)acme
Operations
hint: add logfmt parser
{filename="/var/log/traefik/traefik.log"} |~ `(?i)acme`
Options
Type: Range
Line limit: 1000
Add query
Query inspector
Logs volume
Logs
Time
Unique labels
Wrap lines
Prettify JSON
Deduplication
None
Exact
Numbers
Signature
Display results
Newest first
Oldest first
Common labels: traefik
Line limit: 1000 (608 returned)
Total bytes processed: 72.0 MB
Download
2023-12-07 13:48:09.012 time="2023-12-07T18:48:08Z" level=debug msg="No ACME certificate generation required for domains [\"
v\"]." ACME CA="https://acme-v02.api.letsencrypt.org/directory" providerName=le.acme routerName=prometheus-https@docker rule
="Host(`
)"
Start of range


Sojourn

But Why

- Automate the boring things
- Lessons learned from previous design
  - Promethues/Loki not well suited for many items
  - Event loops are far more flexible
- Inspired by Nemesis and the accompanying blog posts from SpecterOps
  - https://github.com/SpecterOps/Nemesis

Automation as Force Amplification

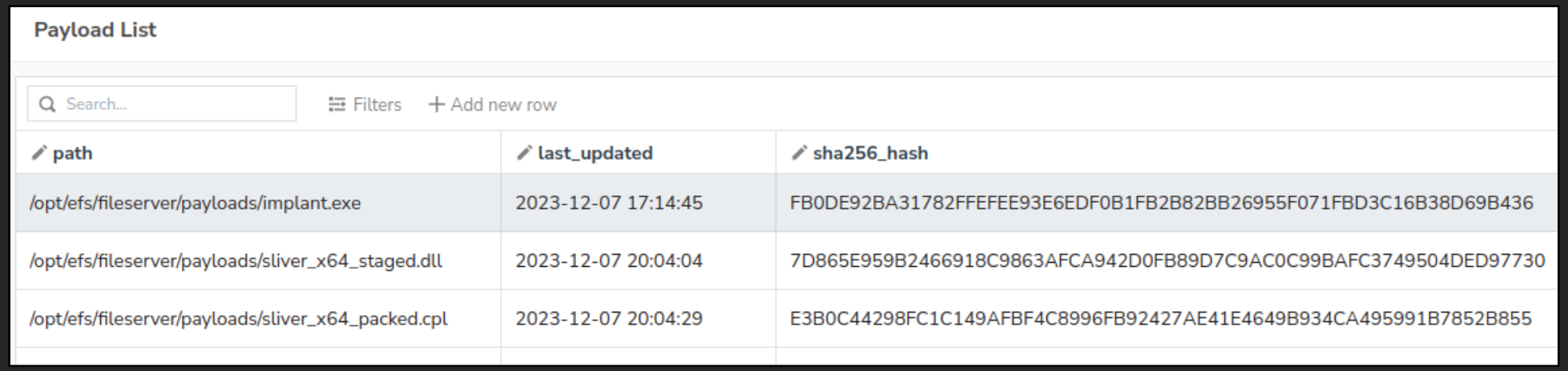
- Allow operators to spend more time flexing their skillset
  - Wrangle large data sets
  - Setup alerts for important events
  - Manage artifacts, both generated and pilfered
  - Log operator activity across various tooling (C2, scanners, phishing, etc)
- Details are abstracted; new automation is dead simple to add

Components

- RabbitMQ: provides pub/sub functionality
- Postgres: general purpose DB for long-term storage
- Core Docker image: provides abstractions for interacting with both data stores
- Appsmith: a low-code UI generator; capable of directly connecting to database providers

Example Use-Case

- The **fs_mon** module publishes **fs_activity** messages
  - Message details include file location and modification timestamp
- The **payload_tracker** module subscribes to **fs_activity** messages
  - If the file is within the “payloads” directory, calculate the hash, store in DB
- The **Payload List** app displays the database data



Other Uses

- *beacon_activity*
  - Generate: monitor C2 servers for operator activity
  - Consume: store within a general-purpose activity log
  - Consume: if it was a download command, parse file for metadata
- *evilginx_callback*
  - Generate: monitor Evilginx for phished creds
  - Consume: start ROADrecon with victim creds or session

Closing Thoughts

Lessons Learned

Traefik is best suited for HTTP

- UDP/TCP routing has significant limitations
- Overlay networks can complicate true source IP preservation

Docker Swarm is limited

- No real support for distributed volumes
- No way to control container creation order

Infrastructure Thoughts

Just because someone did it one way doesn’t mean you have to

Look for ways to simplify and reduce stress in your life (with infra)

Explore new technologies and tools

Be a hacker :)

Resources

https://grafana.com/docs/loki/latest/getting-started/

https://doc.traefik.io/traefik/

https://docs.docker.com/engine/swarm/


Questions?

Example Implementation

https://github.com/SecurityRiskAdvisors/GetInTheBox