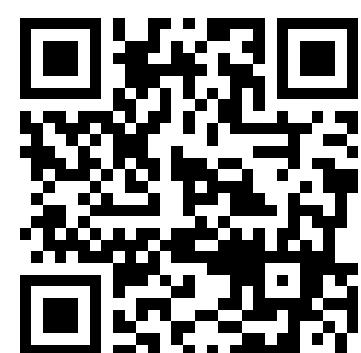


# Le Seigneur Des Conteneurs

Un atelier de migration vers Kubernetes et Traefik



<https://containous.github.io/slides/devoxx-fr-2019>

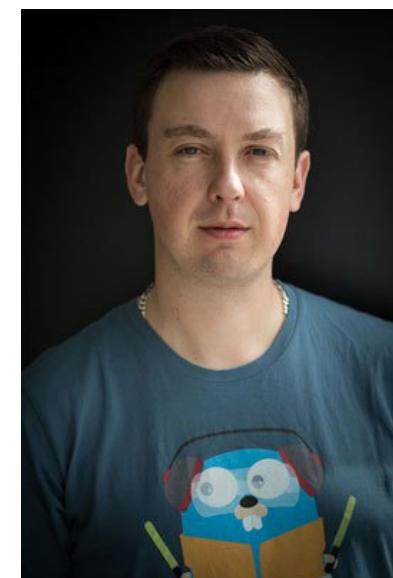
# How To Use These Slides?

- **Browse the slides:** Use the arrows
  - Change chapter: Left/Right arrows
  - Next or previous slide: Top and bottom arrows
- **Overview of the slides:** keyboard's shortcut "o"
- **Speaker mode (and notes):** keyboard's shortcut "s"

# Whoami 1/2

Nicolas Mengin

- DevOps & Code Craftsman @ Containous
- Blacksmith on Traefik
-  @nicomengin
-  nmengin



# Whoami 2/2

Damien DUPORTAL

- Traefik's Developer  Advocate @ Containous
-  @DamienDuportal
-  dduortal



# Containous

<https://containo.us>

- We Believe in Open Source
- We Deliver Traefik
- Commercial Support for Traefik
- 20 people, 90% tech



# Forge Content

- TODO: indicate the checklist
- ....
- ....

*Once Upon A Time...*

# An Infrastructure War

- Docker as a standard
- Orchestrators: Docker Swarm, Rancher Caddle, Mesos Marathon, Kubernetes...
- The war lasted a couple of years...

# *One Orchestrator To Rule Them All*

- **Kubernetes**
- Used by the competition
- Standard in the industry
- Powerful but not easy to master

# Menu

The Hobbit House: Introduction to Traefik with Docker

*Break*

Saruman Tower: Migrate Traefik to Kubernetes

*Break*

The Castle: Migrate the infrastructure to Kubernetes

# The Hobbit House



# The Hobbit House

We want a server in Google Cloud:

- to host our own SCM Server,
- and our own Continuous Integration,
- and a static web site,
- and a "web" command line.

# Infrastructure Setup

- A Google Cloud VM reachable from a public IP  
35.178.178.237 and SSH access
- A domain name lab01.demo.containous.cloud pointing to  
35.178.178.237

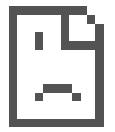
```
$ dig +short lab01.demo.containous.cloud  
35.178.178.237
```

- Docker and docker-compose installed on both the server and the client machines

# Reality Check

<http://lab01.demo.containous.cloud/>

---



This site can't be reached

refused to connect.

Search Google for

ERR\_CONNECTION\_REFUSED

# Lab 1

- Traefik
- Web Server
- CI Server
- SCM: A Gitea Git Server
- Web CLI
- SSL for everyone

# Why Traefik?



Why, Mr Anderson?

# Evolution Of Software Design



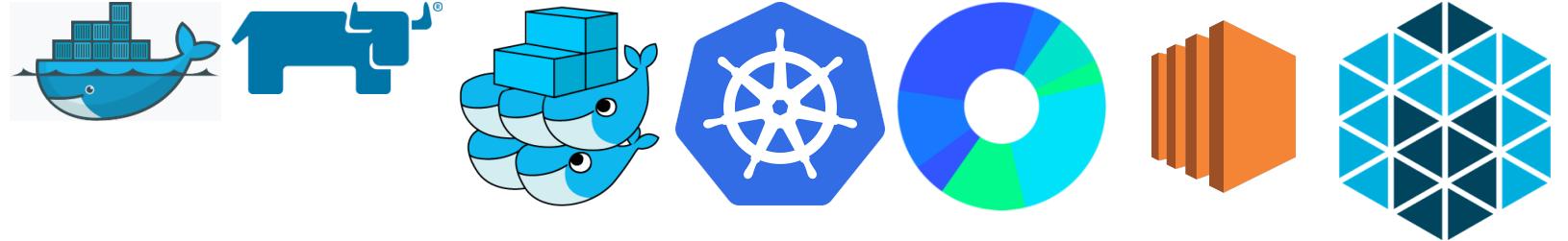
# The Premise Of Microservices...



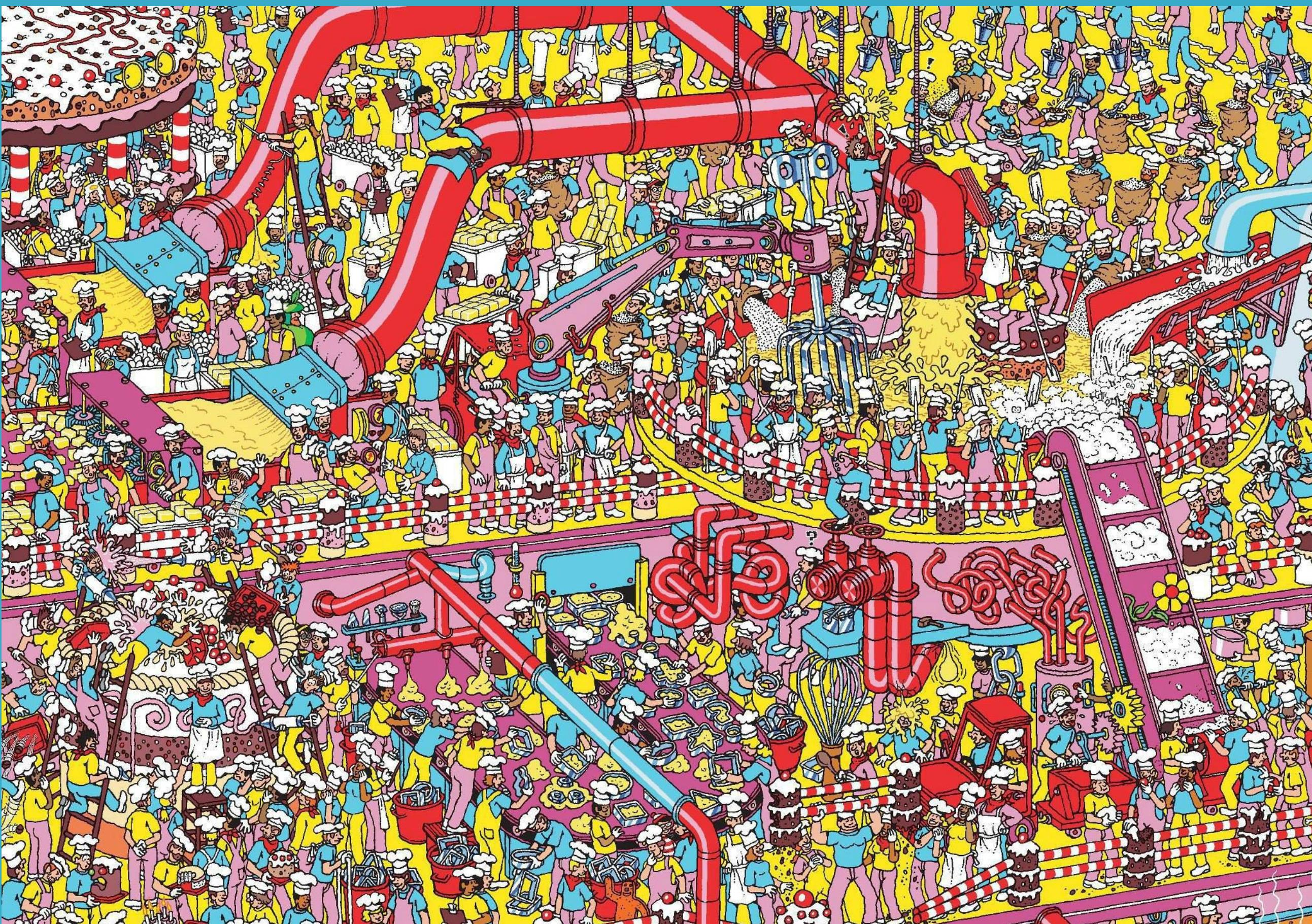
*...And What Happens*

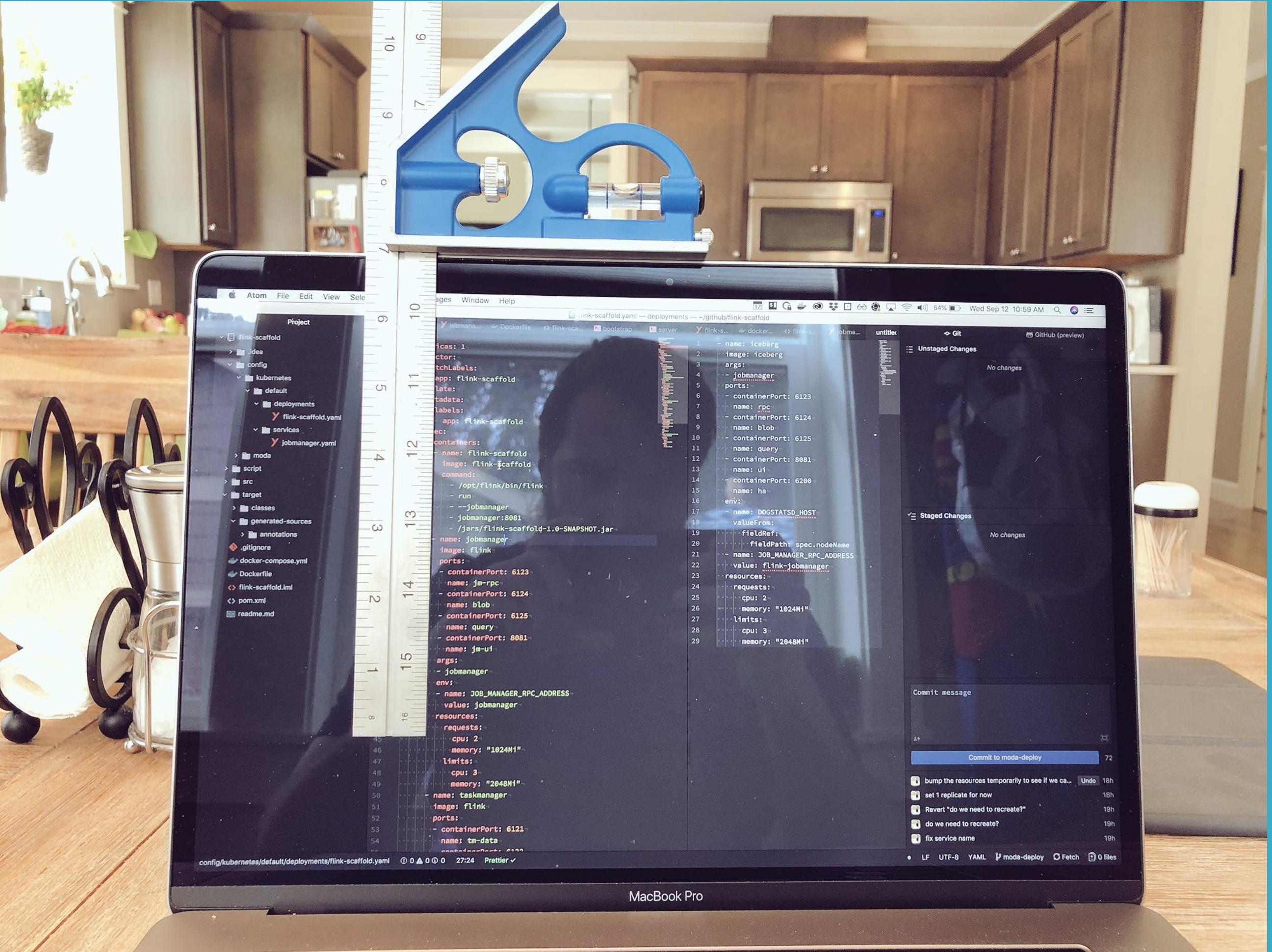


# Tools Of The Trade



# Where's My Service?





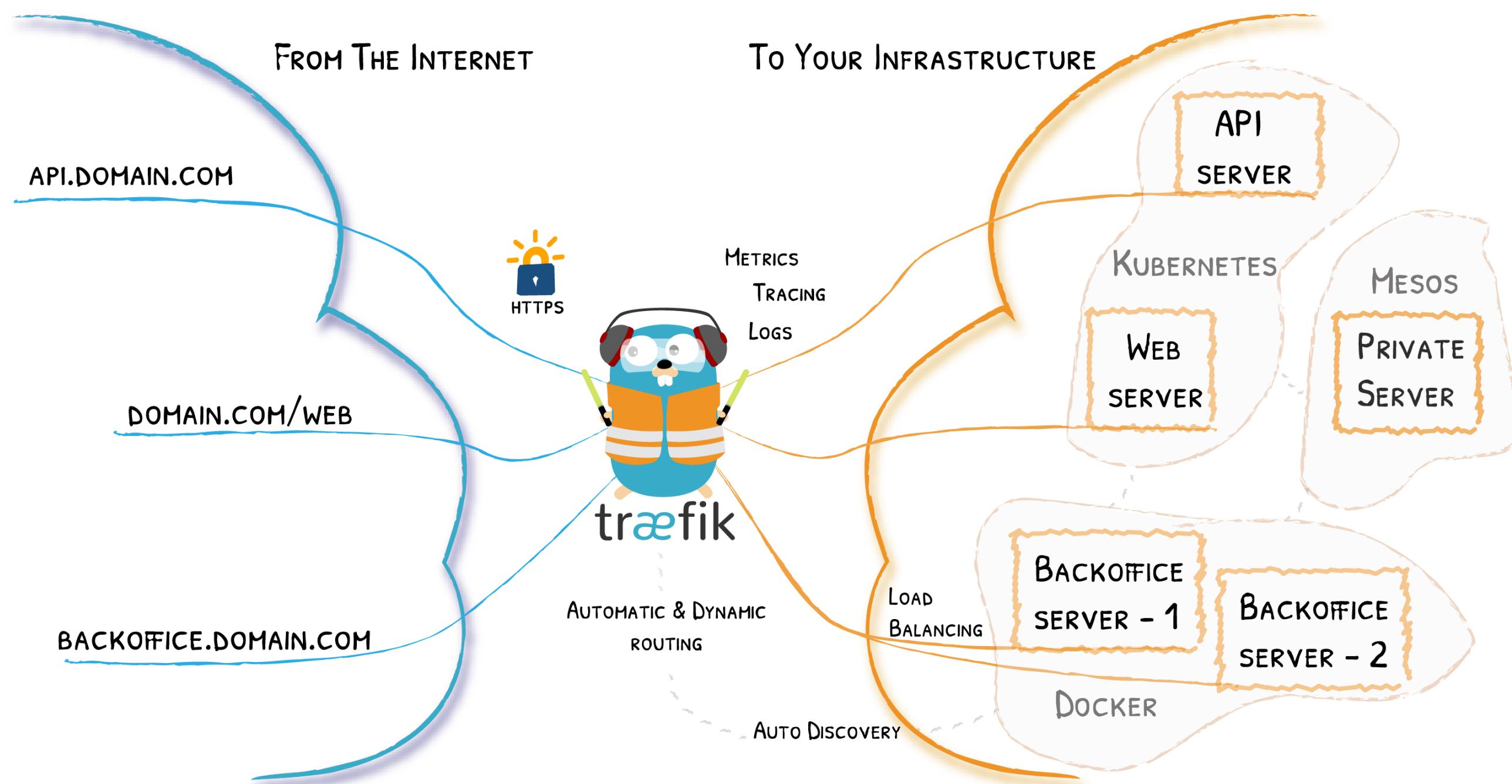
Source: <https://twitter.com/Caged/status/1039937162769096704>

# What If I Told You?



That You Don't Have to Write This Configuration File...?

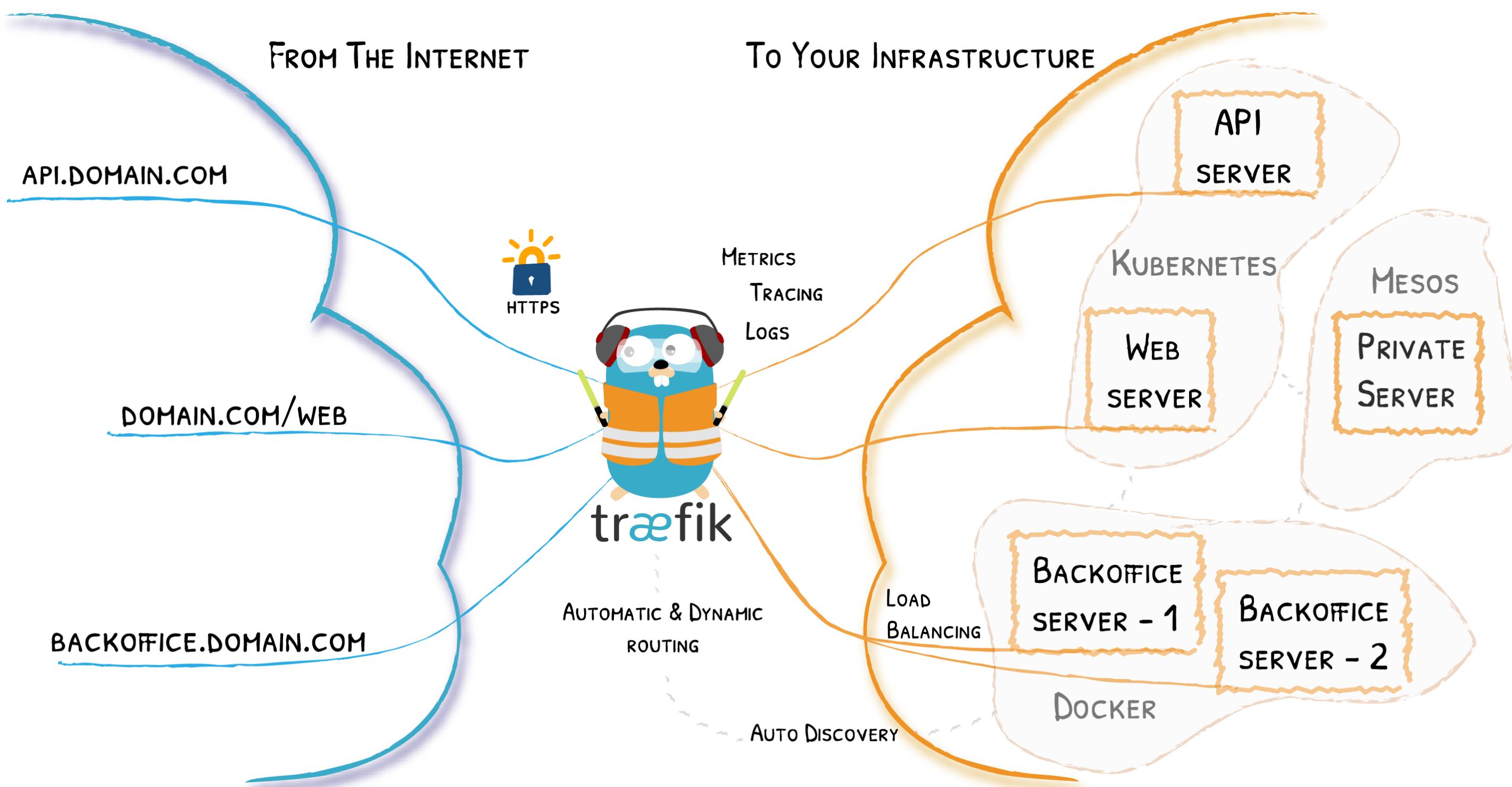
# Here Comes Traefik!



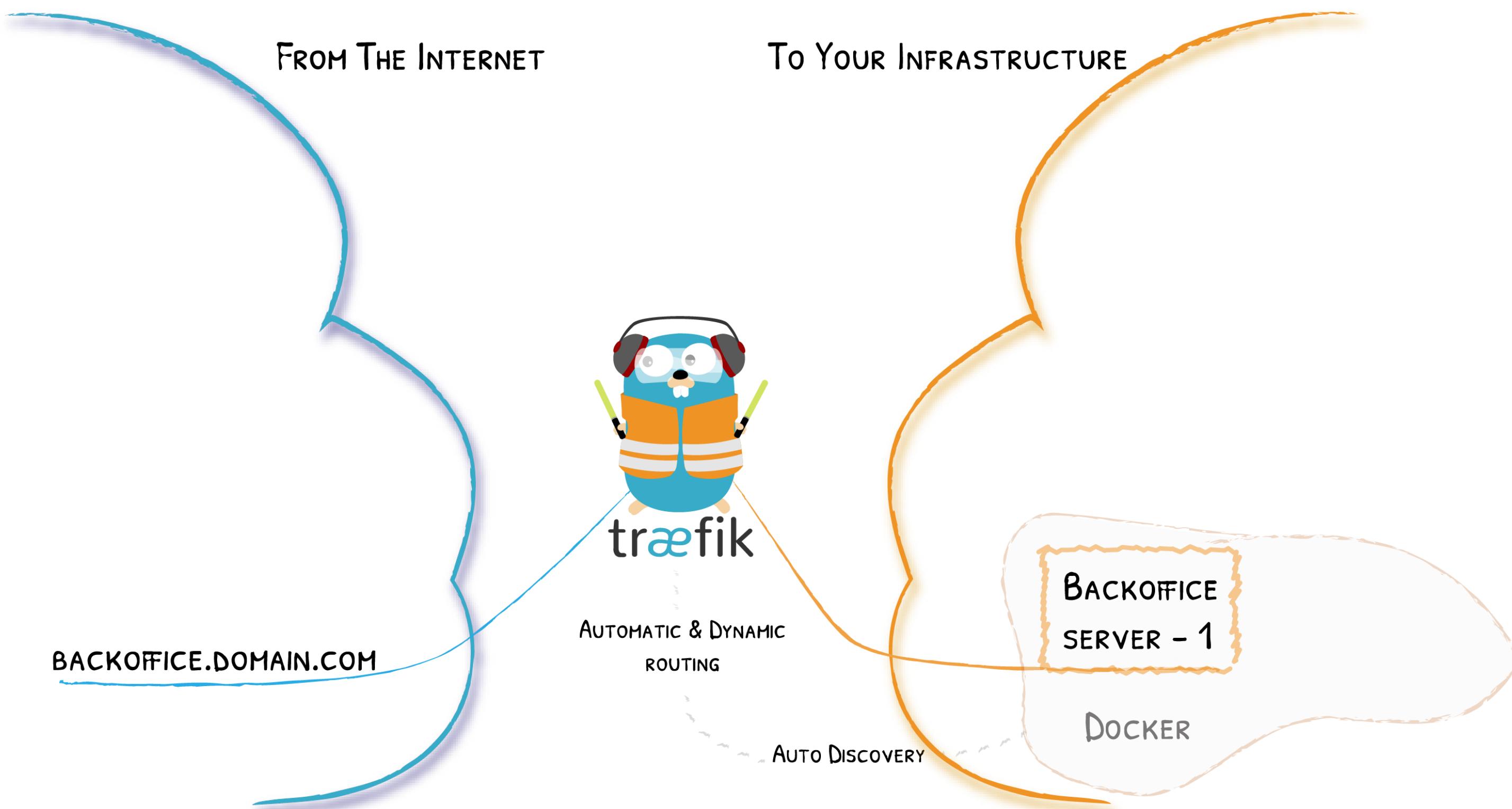
# Traefik Project

-  <https://github.com/containous/traefik>
- MIT License
- Written in Go
- 21,000+ 
- 600M+ 
- 350+ 

# Remember The Diagram?



# Let's Simplify



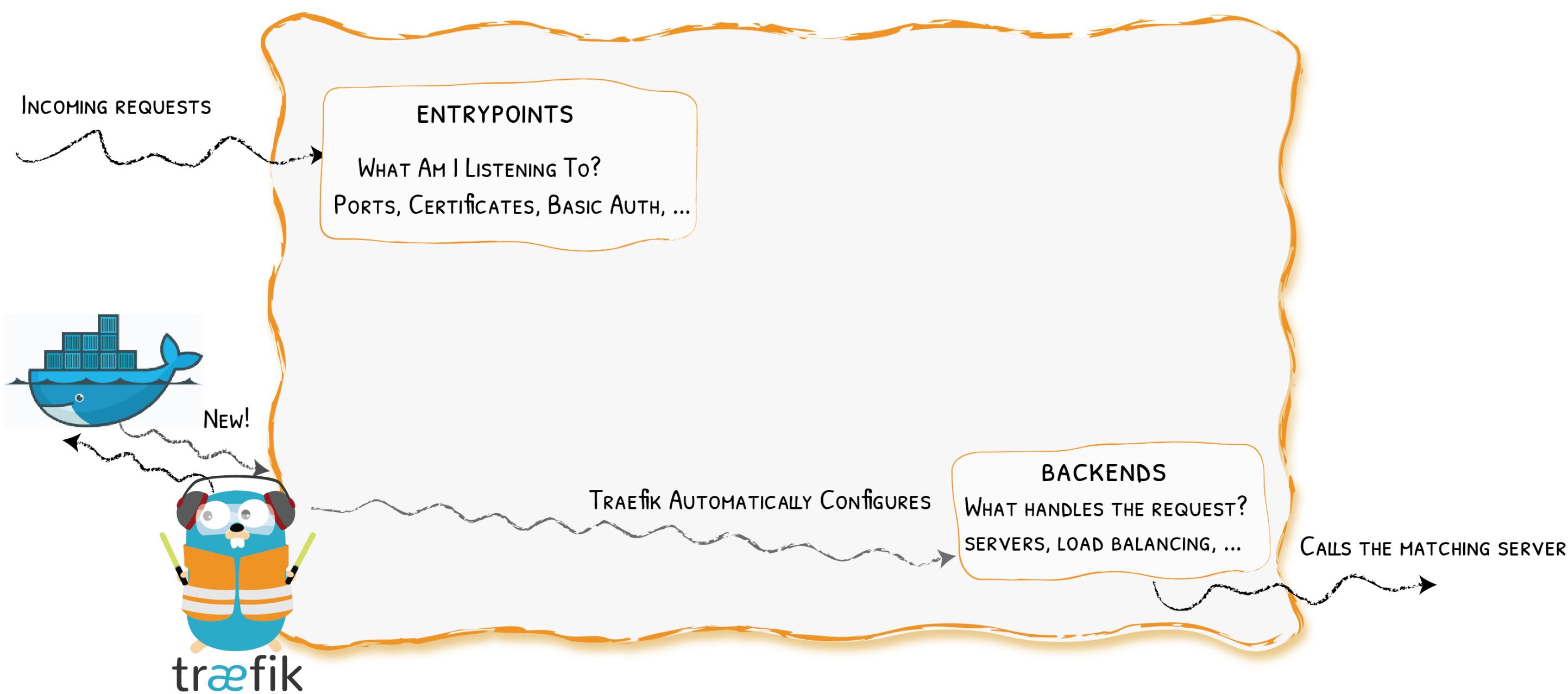
# Providers



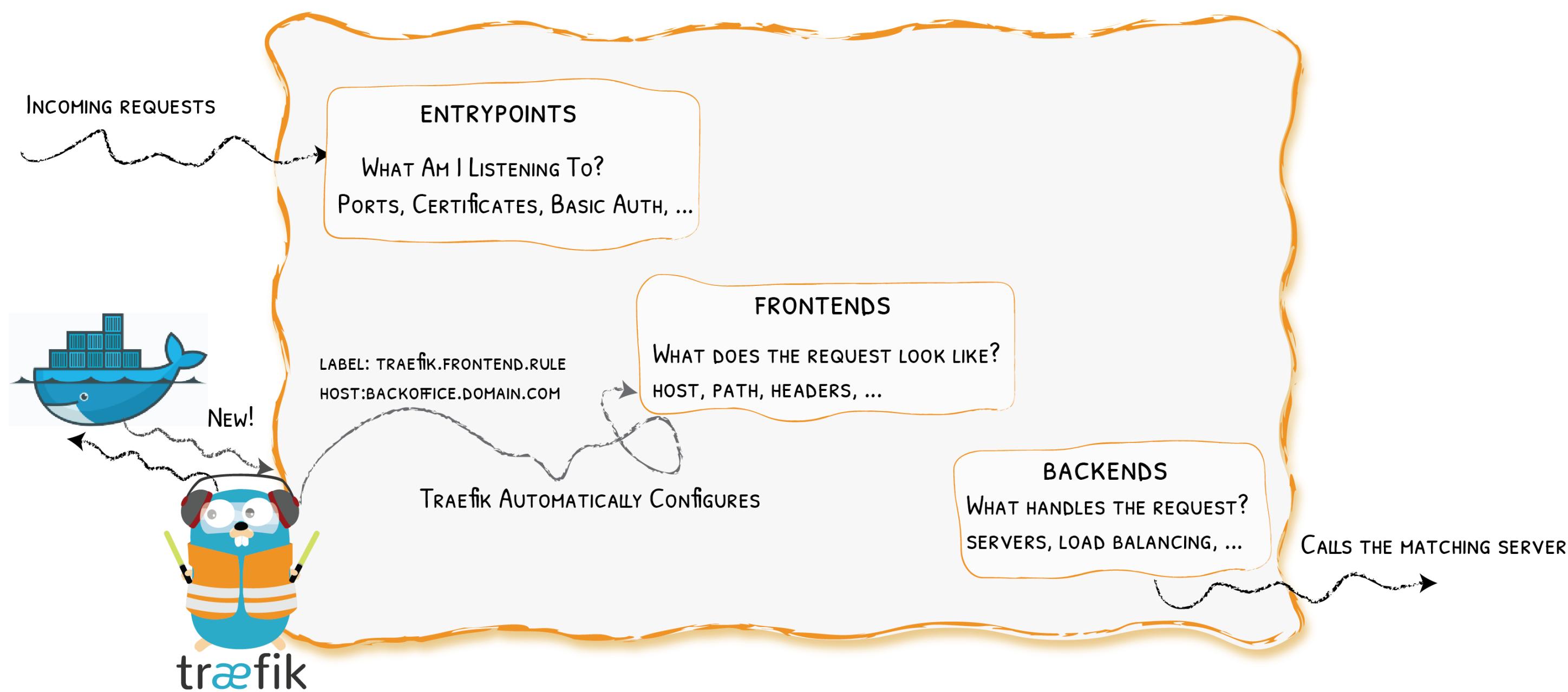
# Entrypoints



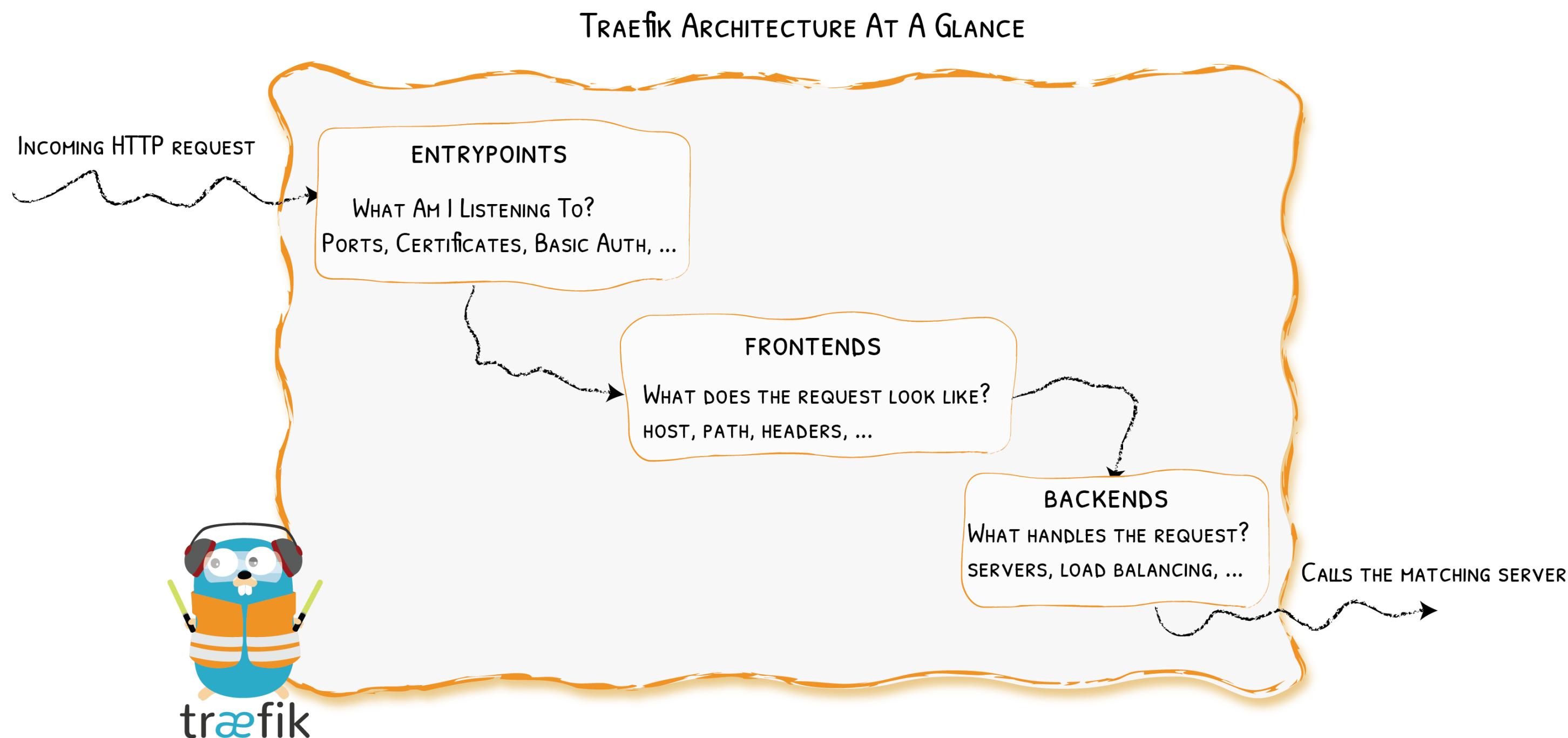
# Backends



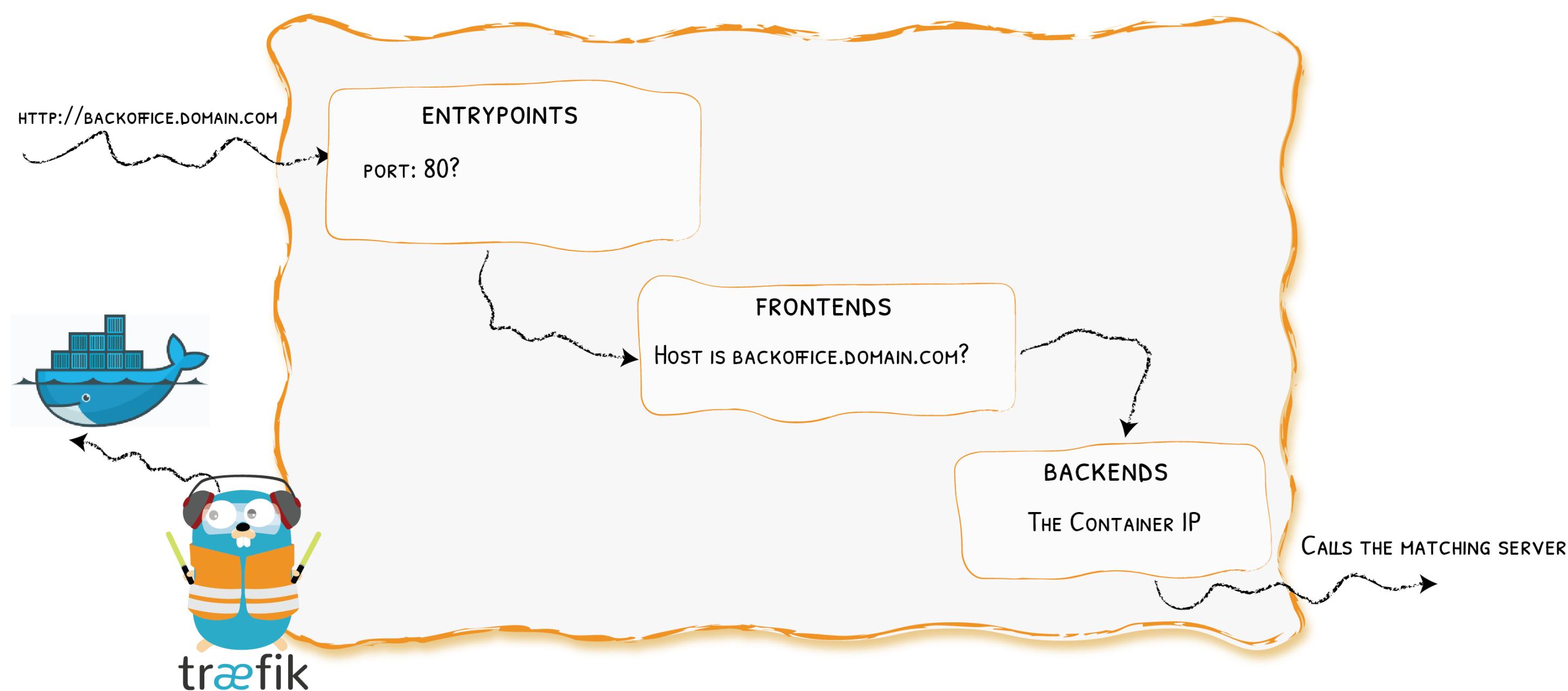
# Frontends



# At A Glance



# In Practice



Let's Go

# Traefik Setup

- Step 1: Compose file in /home/ubuntu/lab-docker-k8s/01-docker/docker-compose.yml:

```
version: '2.4'

services:
  edge:
    image: traefik:1.7.10
    command:
      - "--docker.domain=lab01.demo.containous.cloud"
    ports:
      - "80:80"
      - "443:443"
    volumes:
      # To communicate with the Docker Engine
      - /var/run/docker.sock:/var/run/docker.sock
```

- Step 2: Start the stack:

```
$ docker-compose up -d
```

# Reality Check

<http://lab01.demo.containous.cloud/>



It's good: we have an HTTP answer!

# Lab 1

- Traefik
- **Web Server**
- CI Server
- SCM: A Gitea Git Server
- Web CLI
- SSL for everyone

# *Goal*

We want to host a static webserver behind Traefik.

# Problem

How to tell Traefik to route requests to the web server?

```
http://lab01.demo.containous.cloud/index.html
-> Traefik
-> http://<Webserver Private IP>/index.html
```

# The Web Server Setup

- Step 1: web server in Compose. Check the labels:

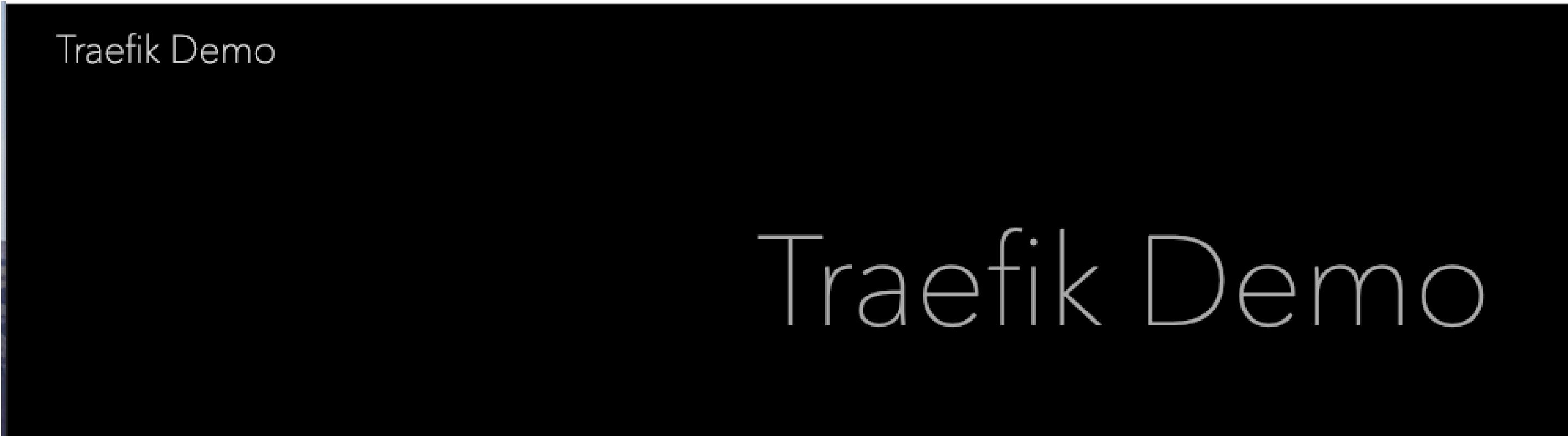
```
web:  
  build: ./web/  
  labels:  
    - "traefik.frontend.rule=Path:/"
```

- Step 2: Start the Web Server:

```
docker-compose up -d web
```

# Reality Check

<http://lab01.demo.containous.cloud/>



It's good: we have a web page!

# Lab 1

- Traefik
- Web Server
- **CI Server**
- SCM: A Gitea Git Server
- Web CLI
- SSL for everyone

# *Goal*

- We want to host our own automation system for Continuous Integration
  - Let's use Jenkins

# Challenge 1/3

- **Problem:** Jenkins exposes 2 ports: 8080 and 50000. How to let Traefik know to only use 8080?
- **Solution:** Select the port with the label `traefik.port`.

```
- "traefik.port=8080"
```

# Challenge 2/3

- **Problem:** How to let Traefik know when to send requests to the Jenkins backend instead of the webserver?

```
http://lab01.demo.containous.cloud/jenkins/configuration  
  -> Traefik  
  -> http://<Jenkins Private IP>:8080/jenkins/configuration
```

- **Solution:** Change the frontend rule to use PathPrefix.

```
- "traefik.frontend.rule=PathPrefix:/jenkins"
```

# Challenge 3/3

- **Problem:** How to tell Jenkins to accept requests under /jenkins?
- **Solution:** Use the Jenkins flag --prefix=/jenkins with the variable JENKINS\_OPTS.

```
environment:  
  - JENKINS_OPTS=--prefix=/jenkins
```

# Jenkins Setup

- Step 1: Edit Compose file:

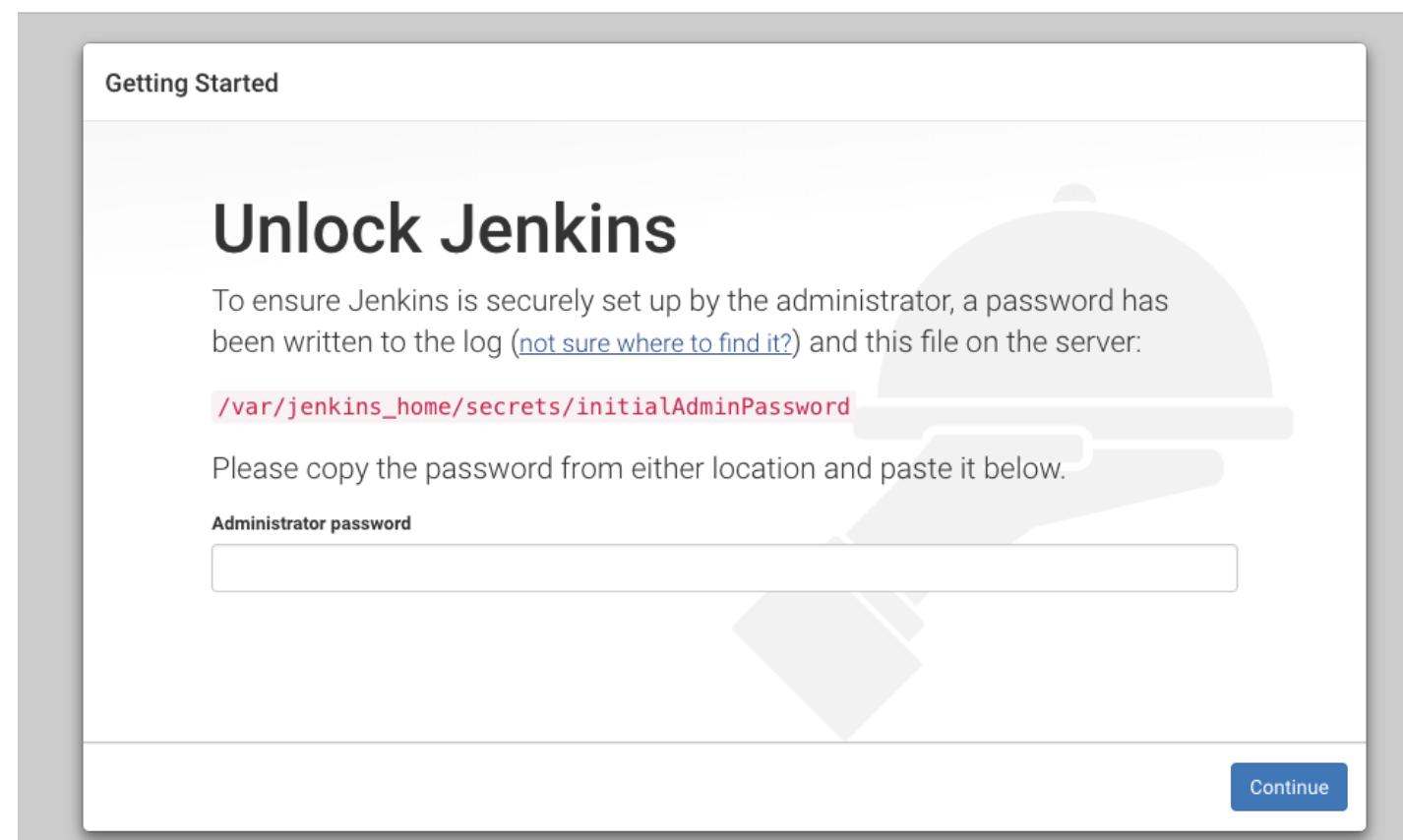
```
jenkins:  
  image: jenkins/jenkins:2.150.3-alpine  
  expose:  
    - 8080  
    - 50000  
  environment:  
    - JENKINS_OPTS=--prefix=/jenkins  
  labels:  
    - "traefik.port=8080"  
    - "traefik.frontend.rule=PathPrefix:/jenkins"
```

- Step 2: Update the service:

```
docker-compose up -d jenkins
```

# Reality Check

<http://lab01.demo.containous.cloud/jenkins>



It's good: we can setup Jenkins!

# Lab 1

- Traefik
- Web Server
- CI Server
- **SCM: A Gitea Git Server**
- Web CLI
- SSL for everyone

# *Goal*

- We want to host our own git server
  - Let's use Gitea, a painless self-hosted Git service.

# Challenge

- **Problem:**
  - Gitea only serves requests under /:
  - How to remove the prefix /gitserver?

```
http://lab01.demo.containous.cloud/gitserver/index.html
  -> Traefik
      -> http://<Gitea private IP>:3000/index.html
```

- **Solution:** Use the Traefik Frontend Rule PathPrefixStrip.

```
- "traefik.frontend.rule=PathPrefixStrip:/gitserver"
```

# Gitea Setup

- Step 1: Edit Compose file:

```
gitserver:  
  image: gitea/gitea:latest  
  expose:  
    - "3000"  
    - "22"  
  environment:  
    - ROOT_URL=/gitserver  
  labels:  
    - "traefik.port=3000"  
    - "traefik.frontend.rule=PathPrefixStrip:/gitserver"
```

- Step 2: Create the service:

```
docker-compose up -d gitserver
```

# Reality Check

<http://lab01.demo.containous.cloud/gitserver>



**Gitea: Git with a cup of tea**

A painless, self-hosted Git service

It's good: we can setup Gitea!

# Lab 1

- Traefik
- Web Server
- CI Server
- SCM: A Gitea Git Server
- **Web CLI**
- SSL for everyone

# Goal

- We want to host our own Web Command Line.
  - Let's use TTYD, Share your terminal over the web.

# Challenge

- **Problem:** TTYD requires Websockets.
- **Solution:** It's not even a problem with Traefik!

# Easy Peasy!

- Step 1: Edit Compose file:

```
ttyd:  
  image: ts10922/ttyd  
  labels:  
    - "traefik.frontend.rule=PathPrefixStrip:/ttyd"
```

- Step 2: Create the service:

```
docker-compose up -d ttyd
```

# Reality Check

<http://lab01.demo.containous.cloud/ttyd>



It's good: we have our own "Dev Box" in a web browser!

# Lab 1

- Traefik
- Web Server
- CI Server
- SCM: A Gitea Git Server
- Web CLI
- **SSL for everyone**

# Goals

- Use HTTPS instead of HTTP
- Do NOT care about certificates and renewal
- Use a TOML configuration file
- Redirect any incoming HTTP request to HTTPS

```
http://lab01.demo.containous.cloud/  
-> https://lab01.demo.containous.cloud/
```



*Let's Encrypt is a free, automated, and open Certificate Authority.*

It uses the "ACME" protocol to verify that you control a given domain name and to issue a certificate.

# Problem 1/4

- **Problem:** How to tell Traefik to listen on port 443 for HTTPS requests?
- **Solution:**
  - Create a new entrypoint
  - Set it as a default entrypoint

```
defaultEntryPoints = ["http", "https"]

[entryPoints]
  [entryPoints.https]
    address = ":443"
    [entryPoints.https.tls]
```

# Problem 2/4

- **Problem:** How to tell Traefik to redirect request from http to https?
- **Solution:** Configure the new entrypoint http.

```
defaultEntryPoints = ["http", "https"]

[entryPoints]
  [entryPoints.https]
    address = ":443"
    [entryPoints.https.tls]
  [entryPoints.http]
    address = ":80"
    [entryPoints.http.redirect]
      entryPoint = "https"
```

# Problem 3/4

- **Problem:** How to tell Traefik to use Let's Encrypt for HTTPS?
- **Solution:**
  - Step 1: Configure the ACME/Let's Encrypt provider:

```
[acme]
email = "lab01@devoxx.lab"
storage = "/acme.json"
entryPoint = "https"
onHostRule = true
```

- Step 2: Give the needed rights to the acme.json file:

```
touch acme.json
chmod 600 acme.json
ls -al
```

# Problem 4/4

- **Problem:**
  - Traefik detects itself as a docker container with a port
  - It tries to request a 2nd certificate for edge.lab01.demo.containous.cloud.
- **Solution:** Exclude Traefik's container with the label traefik.enable=false.

# Traefik Setup 1/2

- Step 1: Create the configuration file `traefik.toml`:

```
defaultEntryPoints = ["http", "https"]

[entryPoints]
  [entryPoints.https]
    address = ":443"
    [entryPoints.https.tls]
  [entryPoints.http]
    address = ":80"
    [entryPoints.http.redirect]
      entryPoint = "https"

[acme]
email = "lab01@devoxx.lab"
storage = "/acme.json"
entryPoint = "https"
onHostRule = true

[docker]
domain=lab01.demo.containous.cloud
watch = true
```

# Traefik Setup 2/2

- Step 2: Adapt the Compose file:

```
version: '2.4'

services:
  edge:
    image: traefik:1.7.10
    labels:
      - "traefik.enable=false"
    ports:
      - "80:80"      # The HTTP port
      - "443:443"    # The HTTPS port
    volumes:
      # To communicate with the Docker Engine
      - /var/run/docker.sock:/var/run/docker.sock
      # Add the TOML configuration file in the root directory
      - ./traefik.toml:/traefik.toml
      # ACME file (mod: 600)
      - ./acme.json:/acme.json
```

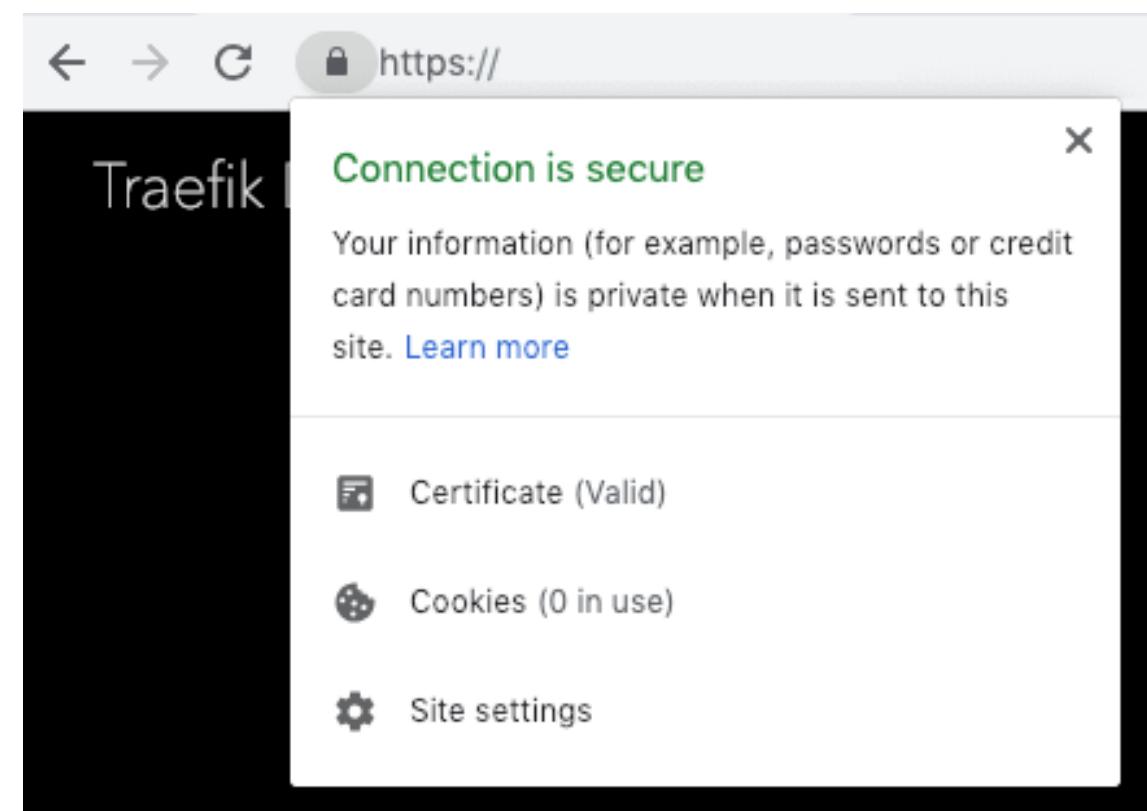
- Step 3: Update the edge service:

```
docker-compose up -d edge
```

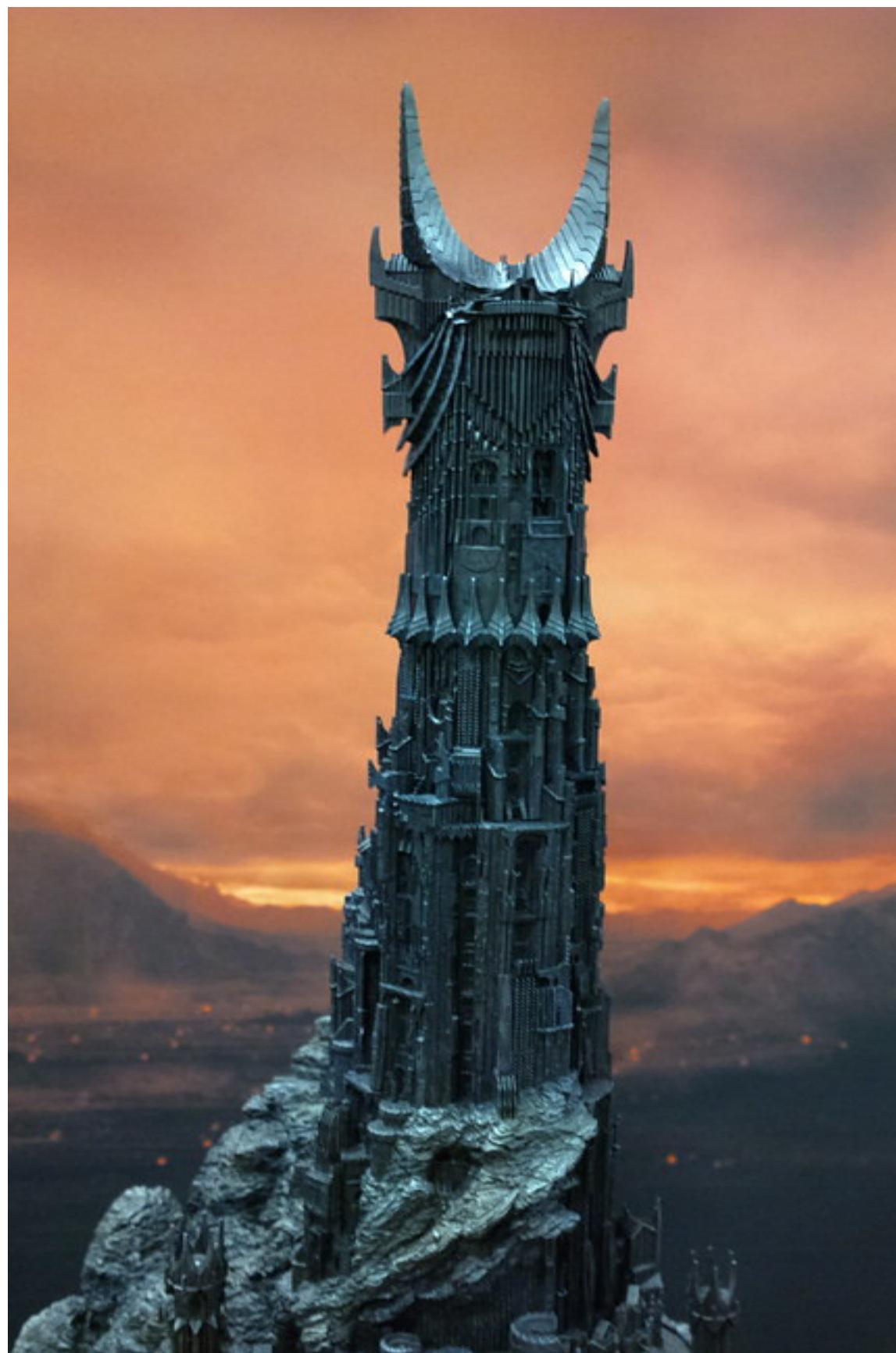
# Reality Check

Wait a few seconds (time to get the certificate from Let's Encrypt) and reload the main page:

<https://lab01.demo.containous.cloud>



# Saruman Tower: Migrate Traefik To Kubernetes



# *Saruman Tower*

We want to begin the migration of our services from the our VM to a Kubernetes cluster:

- keep the Docker services
- migrate Traefik to Kubernetes
- migrate the Let's Encrypt certificates
- access to the Docker services through Traefik in Kubernetes

# *Infrastructure Setup*

- A K3S cluster on a VM reachable from a public IP  
35.178.178.237
- kubectl and helm installed on the client machines

# Lab 2

- Let's Encrypt Certificates
- Traefik
- Web Server
- CI Server
- SCM: A Gitea Git Server
- Web CLI

# *Goal*

- We want to use our generated Let's Encrypt certificates.
  - We do not want to exceed the ACME Rate Limits..

# *Lab To Define*

TODO: dduportal

# Lab 2

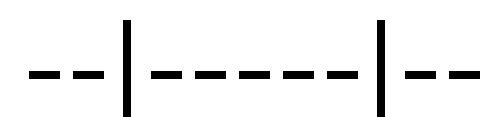
- Let's Encrypt Certificates
- **Traefik**
- Web Server
- CI Server
- SCM: A Gitea Git Server
- Web CLI

# North-South Connection In Kubernetes

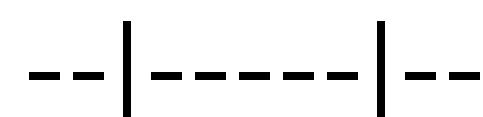
Internet



[ Ingress ]

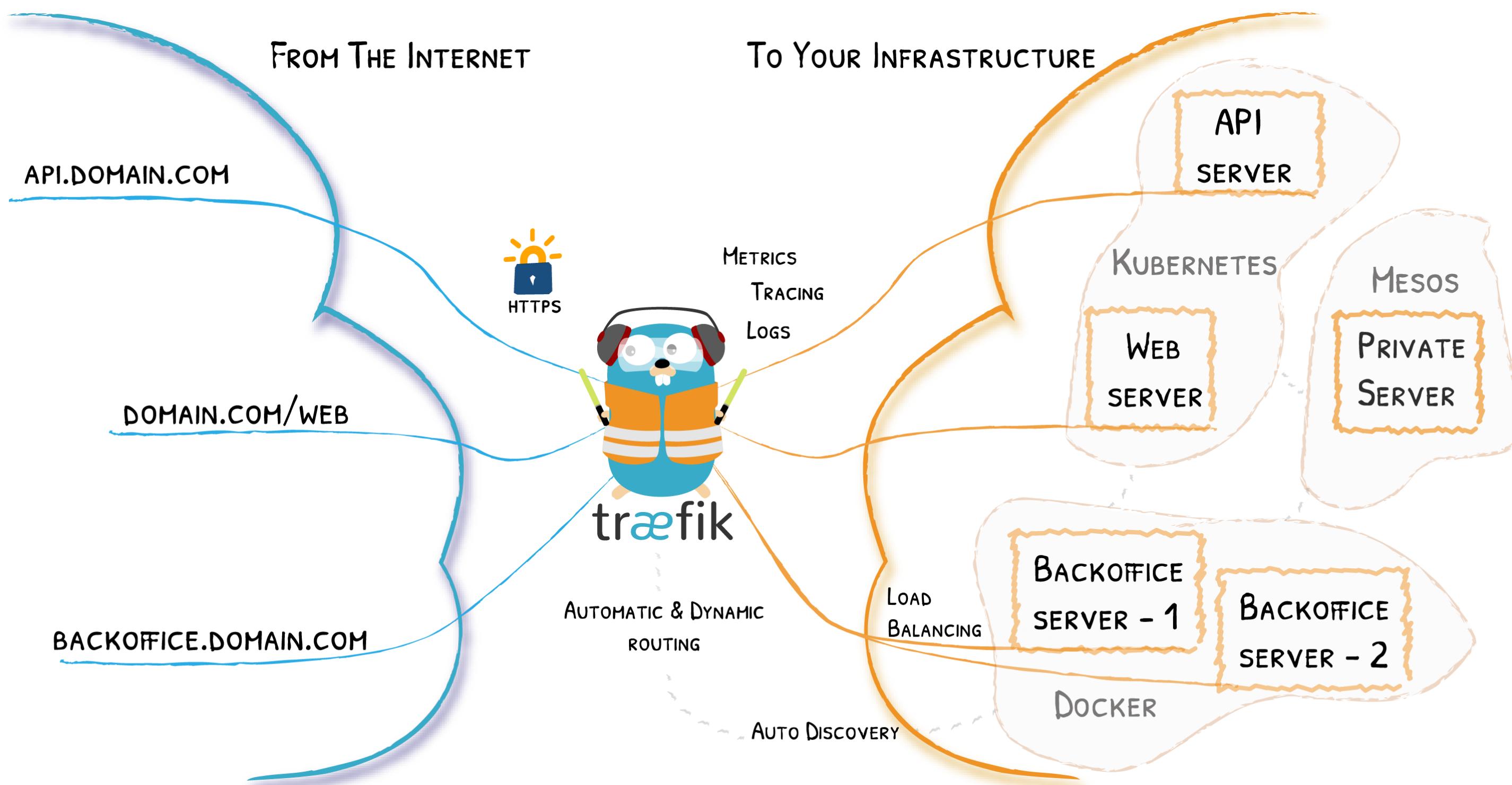


[ Services ]



[ Pods ]

# Remember The Diagram?



# In Kubernetes

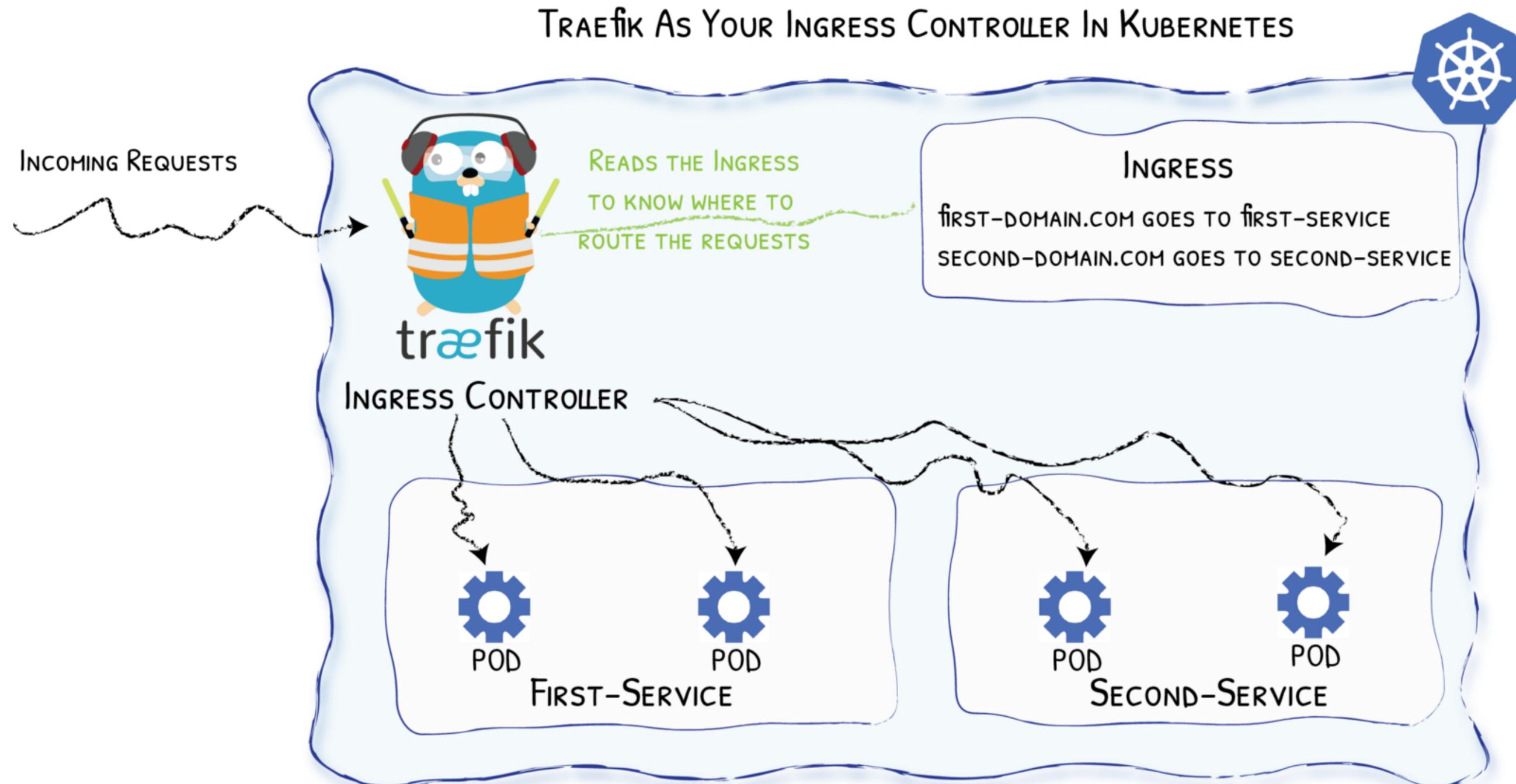


Diagram from <https://medium.com/@geraldcroes>

Let's Go

# Values.Yaml File

- Step 1: Add rights on namespace:

```
# Allow creating the needed Role and Service Account
rbac:
  enabled: true
```

- Step 2: Set SSL EntryPoint with redirection:

```
ssl:
  enabled: true
  enforced: true
```

- Step 3: Add Let's Encrypt:

```
acme:
  enabled: true
  email: devoxxlab@containo.us
  onHostRule: true
  #staging: true
  challengeType: tls-alpn-01
  #  persistence:
  #    existingClaim:
```

# Deploy Traefik

```
helm install stable/traefik \  
--name traefik-devoxx \  
--namespace devoxx \  
--set imageTag=1.7.10 \  
--values values.yml  
helm ls
```

# Access To Traefik

- Step 1: Run the command:

```
kubectl --namespace=devoxx get services
```

- Step 2: Get the Load Balancer IP:

**TODO: dduportal**

- Step 3: Blue-Green on Traefik instances:

**TODO: dduportal**

# Reality Check

<http://lab01.demo.containous.cloud/>



It's good: we have an HTTP answer!

# Lab 2

- Let's Encrypt Certificates
- Traefik
- **Web Server**
- CI Server
- SCM: A Gitea Git Server
- Web CLI

# *Goal*

We want access to the webserver hosted in Docker through Traefik in Kubernetes.

# Challenge 1/2

**Problem:** How to tell to Traefik to route requests to the web server which is not deployed in Kubernetes?

```
https://lab01.demo.containous.cloud/index.html
-> Traefik Kubernetes
-> Traefik Docker
-> https://<Webserver Private IP>/index.html
```

# Headless Service

**Solution:** Use a service linked to an external address.

```
----  
apiVersion: v1  
kind: Service  
metadata:  
  name: web-service  
  namespace: devoxx  
  labels:  
    guilde: web  
spec:  
  ports:  
    # Define the port to contact on the external Host  
    # Here contact Traefik defined in lab1  
    - port: 80  
      name: traefik-http  
    # Indicate to Kubernetes that the service will redirect  
    # to a backend which is not managed in the Kuberntes network  
  type: ExternalName  
  # IP of the VM in the lab1  
  externalName: 18.196.121.238
```

# *Challenge 2/2*

**Problem:** How to detect the HTTPS requests to catch?

# Ingress Rule

**Solution:** Catch all the incoming requests for the Path: /.

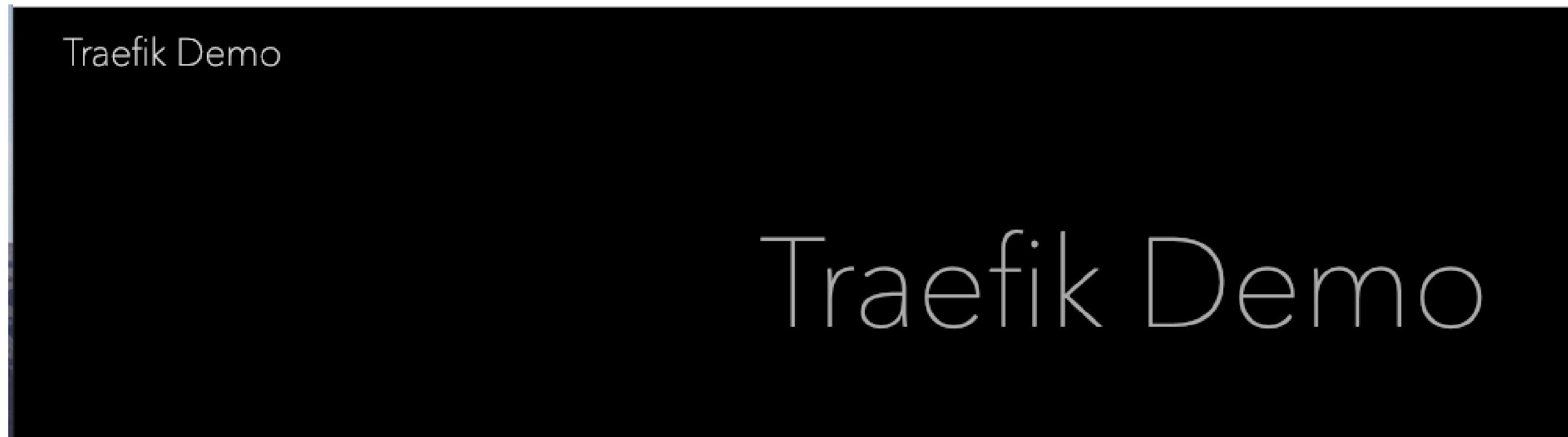
```
----  
apiVersion: extensions/v1beta1  
kind: Ingress  
metadata:  
  name: web-ingress  
  namespace: devoxx  
  labels:  
    guilde: web  
  annotations:  
    kubernetes.io/ingress.class: 'traefik'  
    traefik.frontend.passHostHeader: "false"  
    traefik.frontend.rule.type: PathPrefix  
spec:  
  rules:  
  - host:  
    http:  
      paths:  
      - path: /  
        backend:  
          serviceName: web-service  
          servicePort: traefik-http
```

# *Apply The Configuration*

```
kubectl apply -f ./web/svc-ingress.yml
```

# Reality Check

<https://lab01.demo.containous.cloud/>



It's good: we have a web page!

# Lab 2

- Let's Encrypt Certificates
- Traefik
- Web Server
- **CI Server**
- SCM: A Gitea Git Server
- Web CLI

# *Goal*

We want access to the CI hosted in Docker through Traefik in  
Kubernetes

# Challenge 1/2

**Problem:** How to tell to Traefik to route requests to the CI which is not deployed in Kubernetes?

```
https://lab01.demo.containous.cloud/jenkins
  -> Traefik Kubernetes
  -> Traefik Docker
  -> https://<Jenkins Private IP>/jenkins
```

# Headless Service

**Solution:** Use (once again) a service linked to an external address.

```
---  
apiVersion: v1  
kind: Service  
metadata:  
  name: jenkins-service  
  namespace: devoxx  
  labels:  
    guilde: ci  
spec:  
  ports:  
  - port: 80  
    name: traefik-http  
  type: ExternalName  
  externalName: 18.196.121.238
```

# *Challenge 2/2*

**Problem:** How to detect the HTTPS requests to catch?

# Ingress Rule

**Solution:** Catch the requests for the Path: /jenkins.

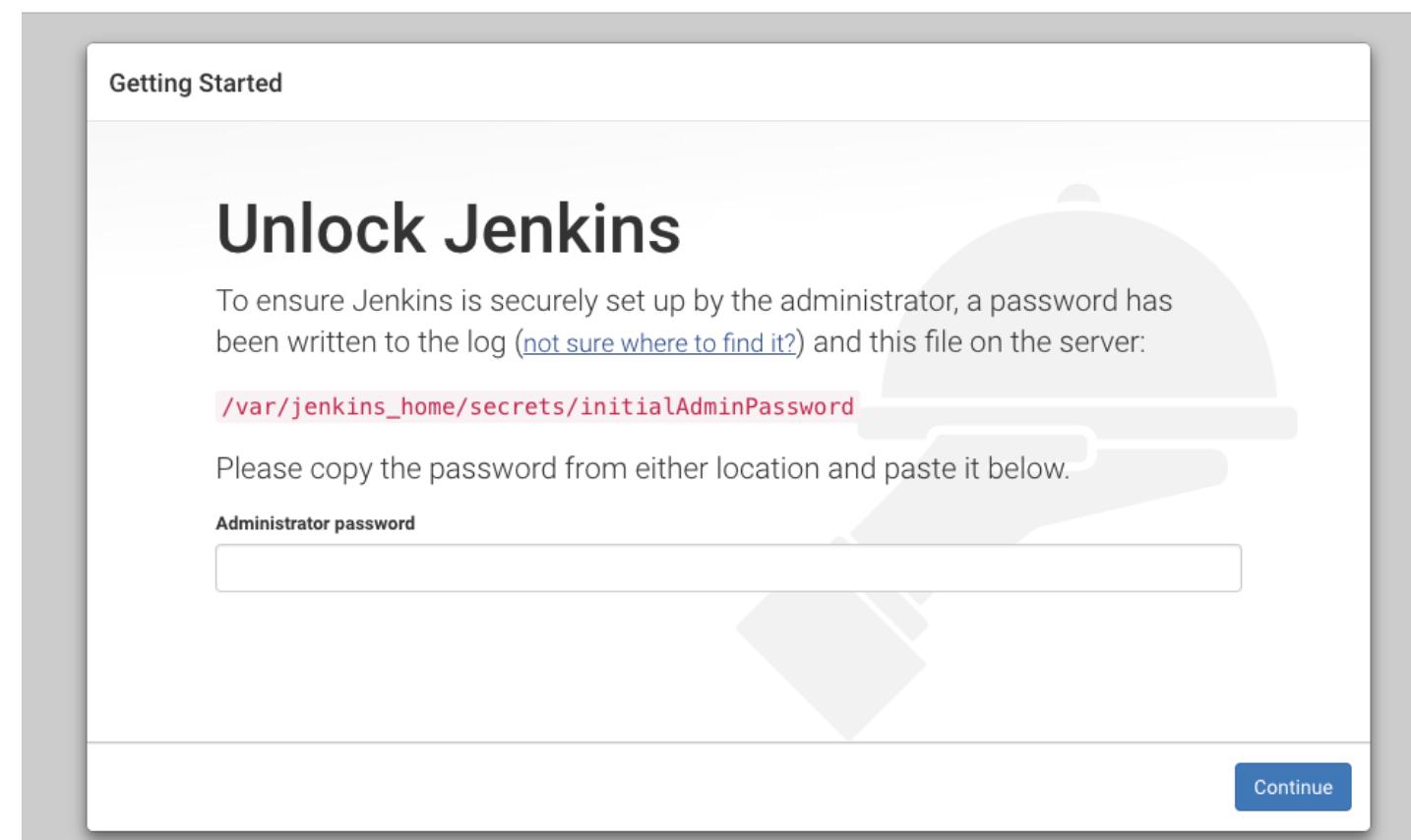
```
---
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: jenkins-ingress
  namespace: devoxx
  labels:
    guilde: ci
  annotations:
    kubernetes.io/ingress.class: 'traefik'
    traefik.frontend.passHostHeader: "false"
    traefik.frontend.rule.type: PathPrefix
spec:
  rules:
  - host:
    http:
      paths:
      - path: /jenkins
        backend:
          serviceName: jenkins-service
          servicePort: traefik-http
```

# *Apply The Configuration*

```
kubectl apply -f ./jenkins/svc-ingress.yml
```

# Reality Check

<https://lab01.demo.containous.cloud/jenkins>



It's good: we still can setup Jenkins!

# Lab 2

- Let's Encrypt Certificates
- Traefik
- Web Server
- CI Server
- **SCM: A Gitea Git Server**
- Web CLI

# *Goal*

We want access to the Git server hosted in Docker through Traefik in Kubernetes.

# Challenge 1/2

- **Problem:**
  - Gitea only serves requests under /
  - Traefik in Docker already removes the prefix /gitserver

```
http://lab01.demo.containous.cloud/gitserver/index.html
  -> Traefik Kubernetes
  -> Traefik Docker
  -> http://<Gitea private IP>:3000/index.html
```

# Use A PathPrefix Rule

**Solution:** Do not remove the prefix (Thanks Captain Obvious!).

```
---
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: gitea-ingress
  namespace: devoxx
  labels:
    guilde: git
  annotations:
    kubernetes.io/ingress.class: 'traefik'
    traefik.frontend.passHostHeader: "false"
    # Only Path Prefix to let the other Traefik Strip it
    traefik.frontend.rule.type: PathPrefix
spec:
  rules:
  - host:
    http:
      paths:
      - path: /gitserver
        backend:
          serviceName: gitea-server
          servicePort: traefik-http
```

# *Challenge 2/2*

**Problem:** How to tell to Traefik to route requests to the SCM which is not deployed in Kubernetes?

# Headless Service

**Solution:** Use (once again again) a service linked to an external address.

```
---  
apiVersion: v1  
kind: Service  
metadata:  
  name: gitea-server  
  namespace: devoxx  
  labels:  
    guilde: git  
spec:  
  ports:  
  - port: 80  
    name: traefik-http  
  type: ExternalName  
  externalName: 18.196.121.238
```

# Reality Check

<https://lab01.demo.containous.cloud/gitserver>



**Gitea: Git with a cup of tea**

A painless, self-hosted Git service

It's good: Gitea is still available!

# Lab 2

- Let's Encrypt Certificates
- Traefik
- Web Server
- CI Server
- SCM: A Gitea Git Server
- **Web CLI**

# *Goal*

We want to access to TTYD deployed in Docker through Traefik in Kubernetes.

# Challenges

- **Problem 1:** How to tell to Traefik to route requests to TTYD which is not deployed in Kubernetes?

```
https://lab01.demo.containous.cloud/ttyd
  -> Traefik Kubernetes
  -> Traefik Docker
  -> https://<WebCLI Private IP>/
```

- **Problem 2:** How to detect the HTTPS requests to catch ?

# Solution 1

- Use a Headless Service:

```
---  
apiVersion: v1  
kind: Service  
metadata:  
  name: ttyd-service  
  namespace: devoxx  
  labels:  
    guilde: console  
spec:  
  ports:  
  - port: 80  
    name: traefik-http  
  type: ExternalName  
  externalName: 18.196.121.238
```

# Solution 2

- Ingress Rule with PathPrefix: /ttyd:

```
---  
apiVersion: extensions/v1beta1  
kind: Ingress  
metadata:  
  name: ttyd-ingress  
  namespace: devoxx  
  annotations:  
    kubernetes.io/ingress.class: 'traefik'  
    traefik.frontend.passHostHeader: "false"  
    traefik.frontend.rule.type: PathPrefix  
spec:  
  rules:  
  - host:  
    http:  
      paths:  
      - path: /ttyd  
        backend:  
          serviceName: ttyd-service  
          servicePort: traefik-http
```

# Reality Check

<https://lab01.demo.containous.cloud/ttyd>



It's good: we can continue to develop in a web browser!

# The Castle



# *The Castle*

We want to terminate the migration of our services to the  
Kubernetes cluster.

# *Infrastructure Setup*

- A K3S cluster on a VM reachable from a public IP  
35.178.178.237
- kubectl and helm installed on the client machines

# Lab 3

- **CI Server**
- SCM: A Gitea Git Server
- Web CLI
- Web Server

# *Goal*

We want to host the CI in Kubernetes and access it through Traefik

# *Challenge 1/3*

**Problem:** How to host the CI in Kubernetes?

# Deployment Object

**Solution:** Deploy it as a Deployment object.

```
---  
kind: Deployment  
apiVersion: extensions/v1beta1  
metadata:  
  name: jenkins-full-deployment  
  namespace: devoxx  
spec:  
  replicas: 1  
  template:  
    metadata:  
      labels:  
        guilde: ci  
        faction: jenkins  
    spec:  
      containers:  
        - name: jenkins-full-container  
          image: jenkins/jenkins:2.150.3-alpine  
          imagePullPolicy: IfNotPresent  
          env:  
            - name: JENKINS_OPTS  
              value: "--prefix=/jenkins"
```

# Challenge 2/2

**Problem:** How to access to the CI?

```
https://lab01.demo.containous.cloud/jenkins  
-> Traefik  
-> https://<Jenkins Private IP>/jenkins
```

# Service

```
----  
apiVersion: v1  
kind: Service  
metadata:  
  name: jenkins-full-service  
  namespace: devoxx  
  labels:  
    guilde: ci  
spec:  
  type: ClusterIP  
  ports:  
    - port: 8080  
      name: jenkins-http  
    - port: 50000  
      name: jenkins-agent  
  selector:  
    guilde: ci  
    faction: jenkins
```

# Ingress Rule

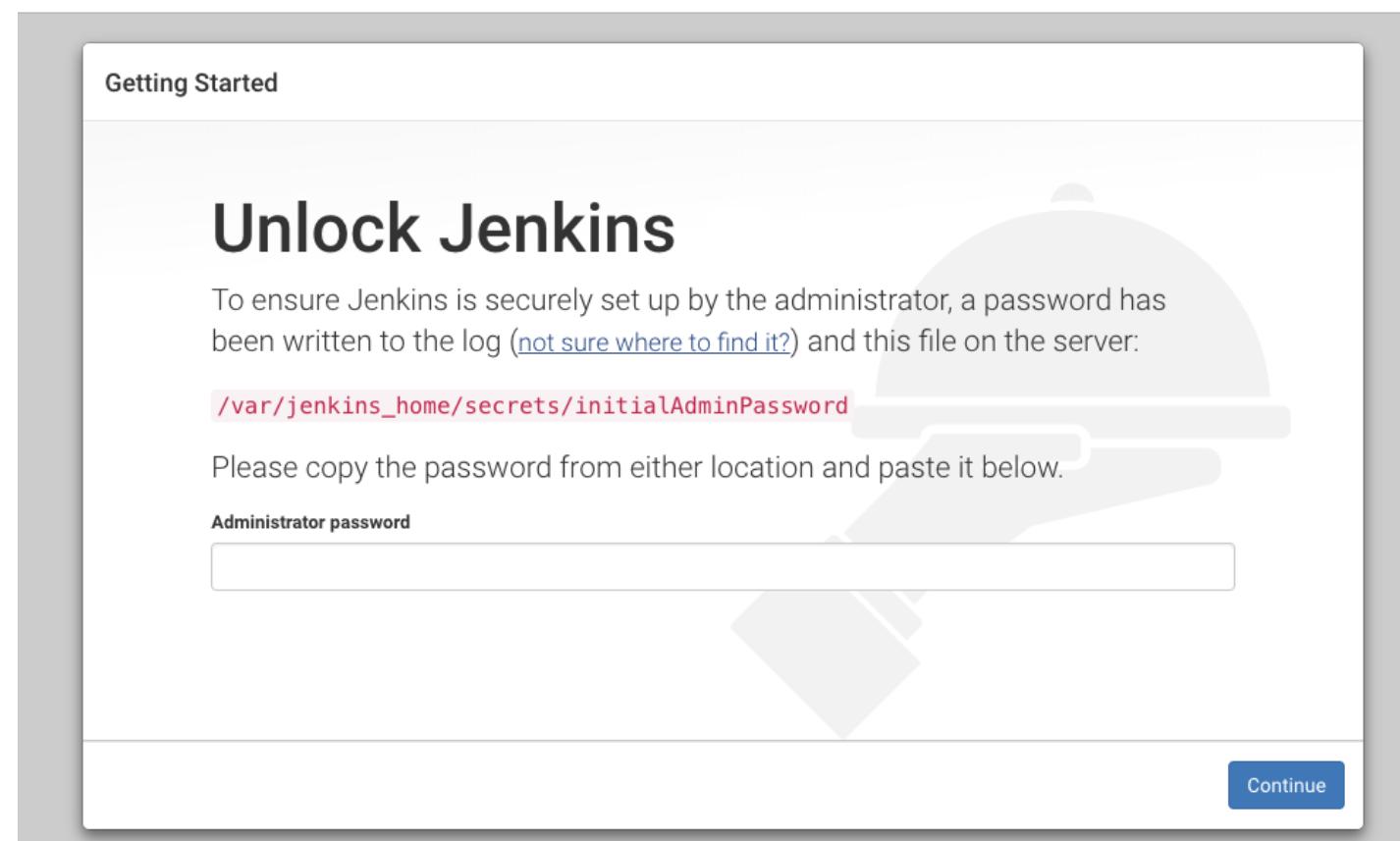
```
---
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: jenkins-full-ingress
  namespace: devoxx
  labels:
    guilde: ci
  annotations:
    kubernetes.io/ingress.class: 'traefik'
    traefik.frontend.rule.type: PathPrefix
spec:
  rules:
  - host:
    http:
      paths:
      - path: /jenkins
        backend:
          serviceName: jenkins-full-service
          servicePort: jenkins-http
```

# Apply The Configuration

```
# Add the new objects
kubectl apply -f ./03-k8s-apps/jenkins/deployment-svc-ingress.yml
# Delete the headless service and its ingress rule (blue-green)
kubectl delete -f ./02-k8s-traefik/jenkins/svc-ingress.yml
```

# Reality Check

<https://lab01.demo.containous.cloud/jenkins>



It's good: we can setup Jenkins in Kubernetes!

# Lab 3

- CI Server
- **SCM: A Gitea Git Server**
- Web CLI
- Web Server

# *Goal*

We want to host the Git server in Kubernetes and access it through Traefik.

# Challenges

- **Problem 1:**
  - How to host the Git server in Kubernetes?
- **Problem 2:**
  - Gitea only serves requests under /
  - Traefik in Docker already removes the prefix /gitserver

```
http://lab01.demo.containous.cloud/gitserver/index.html
-> Traefik
-> http://<Gitea private IP>:3000/index.html
```

# Deployment Object

```
----  
kind: Deployment  
apiVersion: extensions/v1beta1  
metadata:  
  name: gitea-full-deployment  
  namespace: devoxx  
spec:  
  replicas: 1  
  template:  
    metadata:  
      labels:  
        guilde: git  
        faction: server  
    spec:  
      containers:  
        - name: gitea-full-container  
          image: gitea/gitea:latest  
          imagePullPolicy: IfNotPresent  
          env:  
            - name: ROOT_URL  
              value: "/gitserver"
```

# Service

```
----  
apiVersion: v1  
kind: Service  
metadata:  
  name: gitea-full-server  
  namespace: devoxx  
  labels:  
    guilde: git  
spec:  
  type: ClusterIP  
  ports:  
    - port: 3000  
      name: gitea-http  
    - port: 22  
      name: gitea-ssh  
  selector:  
    guilde: git  
    faction: server
```

# Ingress Rule

```
---
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: gitea-full-ingress
  namespace: devoxx
  labels:
    guilde: git
  annotations:
    kubernetes.io/ingress.class: 'traefik'
    traefik.frontend.rule.type: PathPrefixStrip
spec:
  rules:
  - host:
    http:
      paths:
      - path: /gitserver
        backend:
          serviceName: gitea-full-server
          servicePort: gitea-http
```

# Apply The Configuration

```
# Add the new objects
kubectl apply -f ./03-k8s-apps/gitea/deployment-svc-ingress.yml
# Delete the headless service and its ingress rule (blue-green)
kubectl delete -f ./02-k8s-traefik/gitea/svc-ingress.yml
```

# Reality Check

<https://lab01.demo.containous.cloud/gitserver>



**Gitea: Git with a cup of tea**

A painless, self-hosted Git service

It's good: we can setup Gitea in Kubernetes!

# Lab 3

- CI Server
- SCM: A Gitea Git Server
- **Web CLI**
- Web Server

# *Goal*

We want to host TTYD in Kubernetes and access it through Traefik.

# Challenges

- **Problem 1:** How to host the TTYD in Kubernetes?
- **Problem 2:** How to access to TTYD?

```
http://lab01.demo.containous.cloud/ttyd/  
  -> Traefik  
      -> http://<WebCLI private IP>/
```

# Deployment Object

```
----  
kind: Deployment  
apiVersion: extensions/v1beta1  
metadata:  
  name: ttyd-full-deployment  
  namespace: devoxx  
spec:  
  replicas: 1  
  template:  
    metadata:  
      labels:  
        guilde: console  
        faction: tty  
    spec:  
      containers:  
        - name: ttyd-full-container  
          image: ts10922/ttyd:latest  
          imagePullPolicy: IfNotPresent
```

# Service

```
----  
apiVersion: v1  
kind: Service  
metadata:  
  name: ttyd-full-service  
  namespace: devoxx  
  labels:  
    guilde: console  
spec:  
  type: ClusterIP  
  ports:  
    - port: 7681  
      name: ttyd-ws  
  selector:  
    guilde: console  
    faction: tty
```

# Ingress Rule

```
----  
apiVersion: extensions/v1beta1  
kind: Ingress  
metadata:  
  name: ttyd-full-ingress  
  namespace: devoxx  
  annotations:  
    kubernetes.io/ingress.class: 'traefik'  
    traefik.frontend.rule.type: PathPrefixStrip  
spec:  
  rules:  
  - host:  
    http:  
      paths:  
      - path: /ttyd  
        backend:  
          serviceName: ttyd-full-service
```

# Apply The Configuration

```
# Add the new objects
kubectl apply -f ./03-k8s-apps/ttyd/deployment-svc-ingress.yml
# Delete the headless service and its ingress rule (blue-green)
kubectl delete -f ./02-k8s-traefik/ttyd/svc-ingress.yml
```

# Reality Check

<https://lab01.demo.containous.cloud/ttyd>



It's good: we have our own "Dev Box" in a web browser hosted in Kubernetes!

# Lab 3

- CI Server
- SCM: A Gitea Git Server
- Web CLI
- **Web Server**

# Goal

- We want to deploy a new version of the webserver:
  - hosted in Kubernetes and accessed through Traefik
  - continue to access to the old version for the main part of the traffic

# Challenge 1/3

**Problem:** How to host the new version of the webserver in Kubernetes?

# Deployment Object

```
----  
kind: Deployment  
apiVersion: extensions/v1beta1  
metadata:  
  name: web-full-deployment  
  namespace: devoxx  
spec:  
  replicas: 1  
  template:  
    metadata:  
      labels:  
        guild: web  
        faction: server  
    spec:  
      containers:  
        - name: web-full-container  
          image: nmengin/web:devoxx  
          imagePullPolicy: IfNotPresent
```

# Challenge 2/2

**Problem:** How to access to both the old and new version at the same time with a traffic repartition?

```
https://lab01.demo.containous.cloud/  
-> Traefik Kubernetes  
-> 80% of traffic:  
    -> Traefik Docker  
    -> https://<Webserver Docker Private IP>/  
-> 20% of traffic:  
    -> https://<Webserver kubernetes Private IP>/
```

# Follow The Yellow Bird!

**Solution:** Use the Traffic splitting feature in Traefik.

```
traefik.ingress.kubernetes.io/service-weights: |
  web-service: 80%
  web-full-service: 20%
```

# Service

```
----  
apiVersion: v1  
kind: Service  
metadata:  
  name: web-full-service  
  namespace: devoxx  
  labels:  
    guilde: web  
spec:  
  type: ClusterIP  
  ports:  
    - port: 80  
      name: web-http  
  selector:  
    labels:  
      guilde: web  
      faction: server
```

# Ingress Rule

```
---  
apiVersion: extensions/v1beta1  
kind: Ingress  
metadata:  
  name: web-full-ingress  
  namespace: devoxx  
  labels:  
    guilde: web  
  annotations:  
    kubernetes.io/ingress.class: 'traefik'  
    traefik.frontend.passHostHeader: "false"  
    traefik.frontend.rule.type: PathPrefix  
    traefik.ingress.kubernetes.io/service-weights: |  
      web-service: 80%  
      web-full-service: 20%  
spec:  
  rules:  
  - host:  
    http:  
      paths:  
      - path: /  
        backend:  
          serviceName: web-full-service
```

# Apply The Configuration

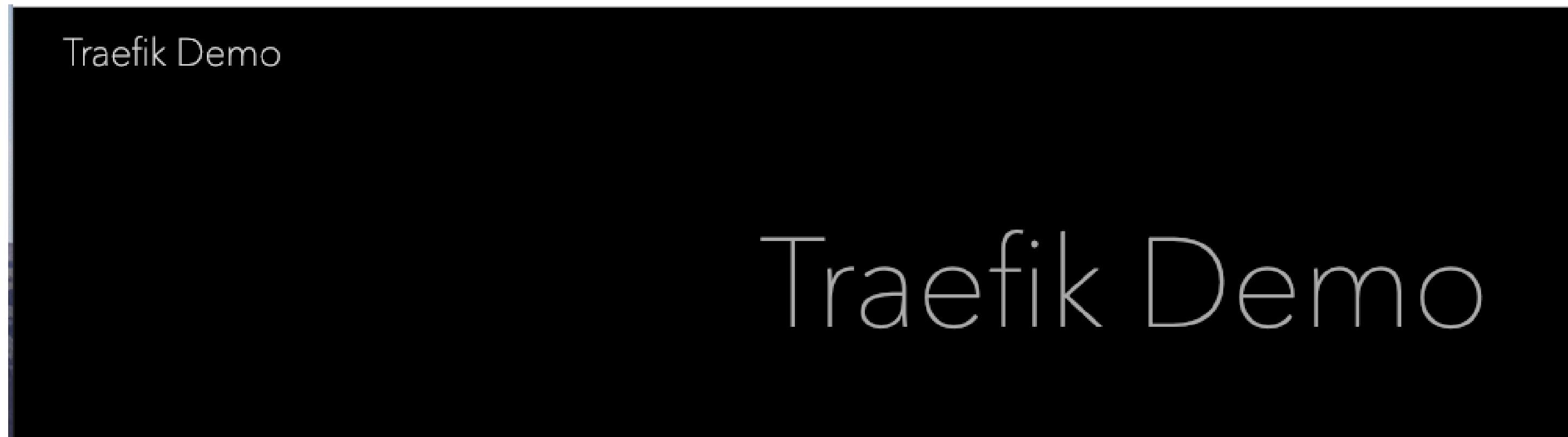
```
# Add the new objects
kubectl apply -f ./03-k8s-apps/jenkins/deployment-svc-ingress.yml
# Delete only the old ingress rule: the service will be reachable from the new one
kubectl --namespace delete ingress web-ingress
```

# Switch All Traffic To The New Version

```
# Edit the ingress
kubectl --namespace devoxx edit ingress web-full-ingress
#####
# Delete the following lines
    traefik.frontend.passHostHeader: "false"
    traefik.frontend.rule.type: PathPrefix
    traefik.ingress.kubernetes.io/service-weights: |
        web-service: 80%
        web-full-service: 20%
...
    - path: /
      backend:
        serviceName: web-service
# Exit and save
###
```

# Reality Check

<https://lab01.demo.containous.cloud/>



It's good: we have a web page in the 2 versions!

We Did Not Talk About...

# Traefik V2

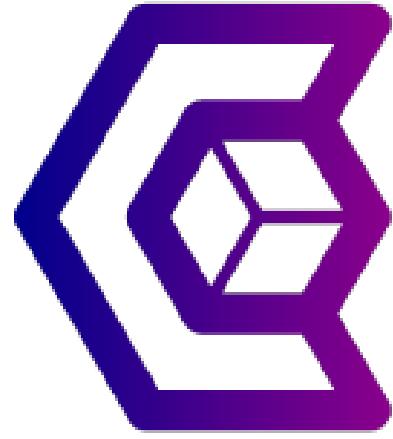
- Used during all the workshop
- Alpha version for a month
- Main features:
  - TCP
  - CRD
  - ...
- Documentation

# High Availability

- TraefikEE
- Split responsibilities
  - Control Plane
  - Data Plane
- Let's Encrypt Distributed Support
- 1.0.0 GA since... Today!
- Documentation



# We Are Hiring!

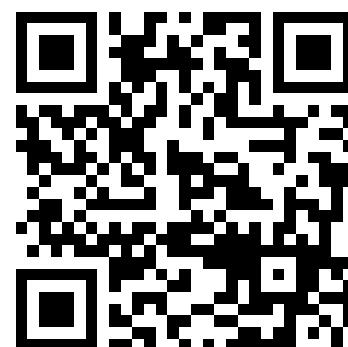


```
docker run -it containous/jobs
```

# Thank You!

🐦 @nicomengin 🐕 nmengin

🐦 @DamienDuportal 🐕 dduportal



- Slides (HTML): <https://containous.github.io/slides/devoxx-fr-2019>
- Slides (PDF): <https://containous.github.io/slides/devoxx-fr-2019/slides.pdf>
- Source on 🐕: <https://github.com/containous/slides/tree/devoxx-fr-2019>