# Traefik Concepts and Configurations

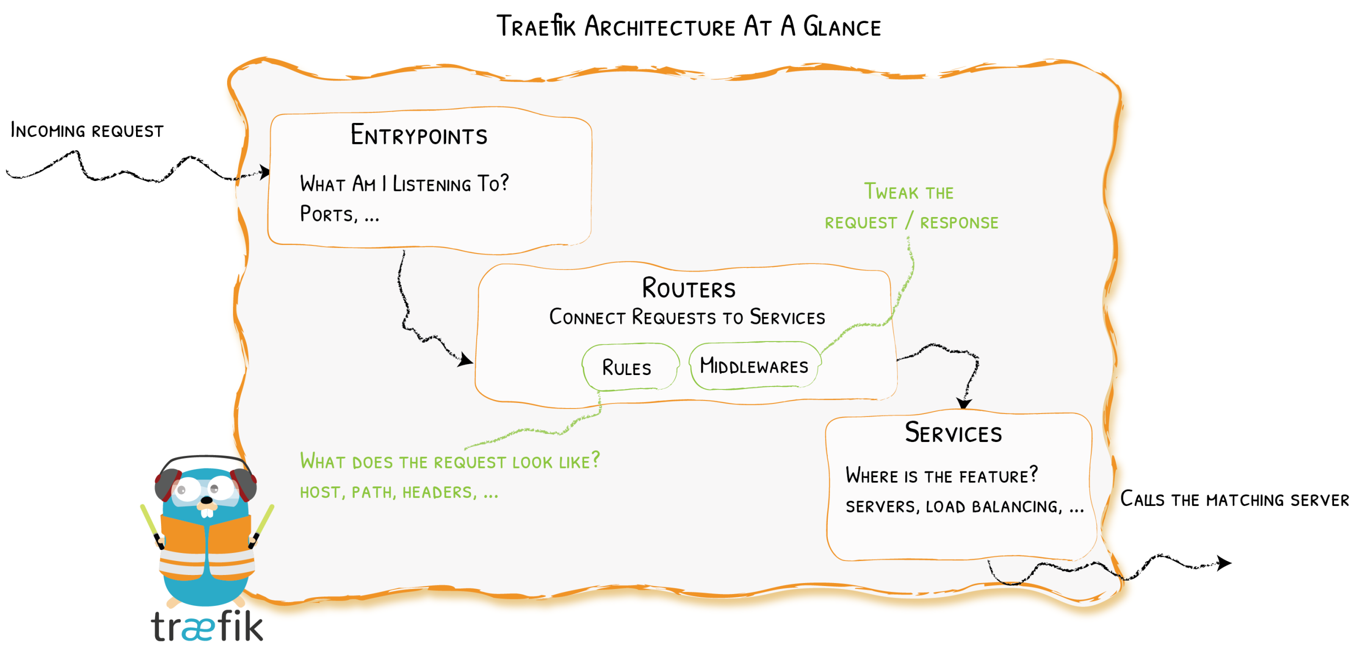
```
traefik is an open source edge router that can detect automatically the right configuration for our services.
basically there are 3 different concepts in traefik architecture.
```

at first, we have entrypoints that actually, are listen ports for external traffics to come in. when the traffics came in to our traefik instances through the ports that are listen, they've passed to

another section called router. this part, is responsible for checking some rules that we set to match with traffics and if those rules are matched, based on our policy we can send that traffic to another

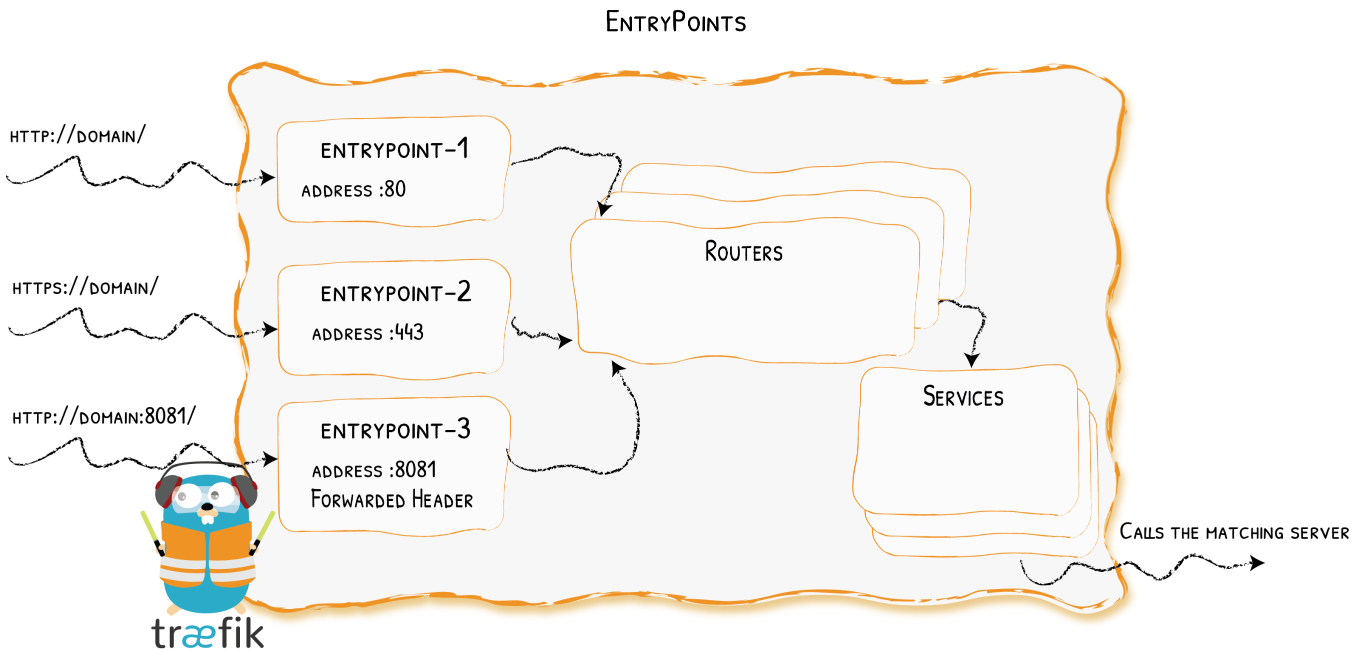
sub-section of router section called middlewares. in middleware we can set some condition and manipulation on traffic to change its condition to our desired state. after all conditions are satisfied,

the traffics has permissions to get to the next step which is our service that points to our real application. in figure below we can see Traefik architecture at a glance.



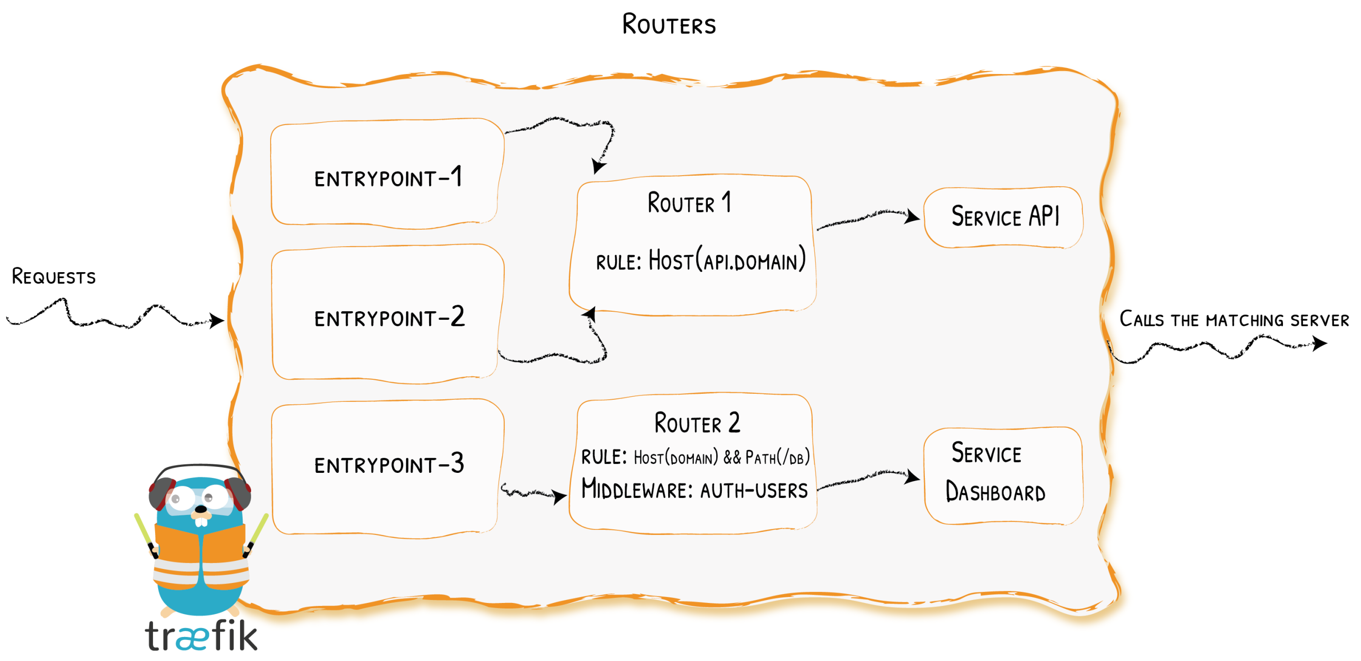
as we told earlier, Entrypoints are first component that incoming traffics can meet. basically those are some ports that are listen from traefik instances to accept incoming requests. the overall structure

of entrypoints are shown below.

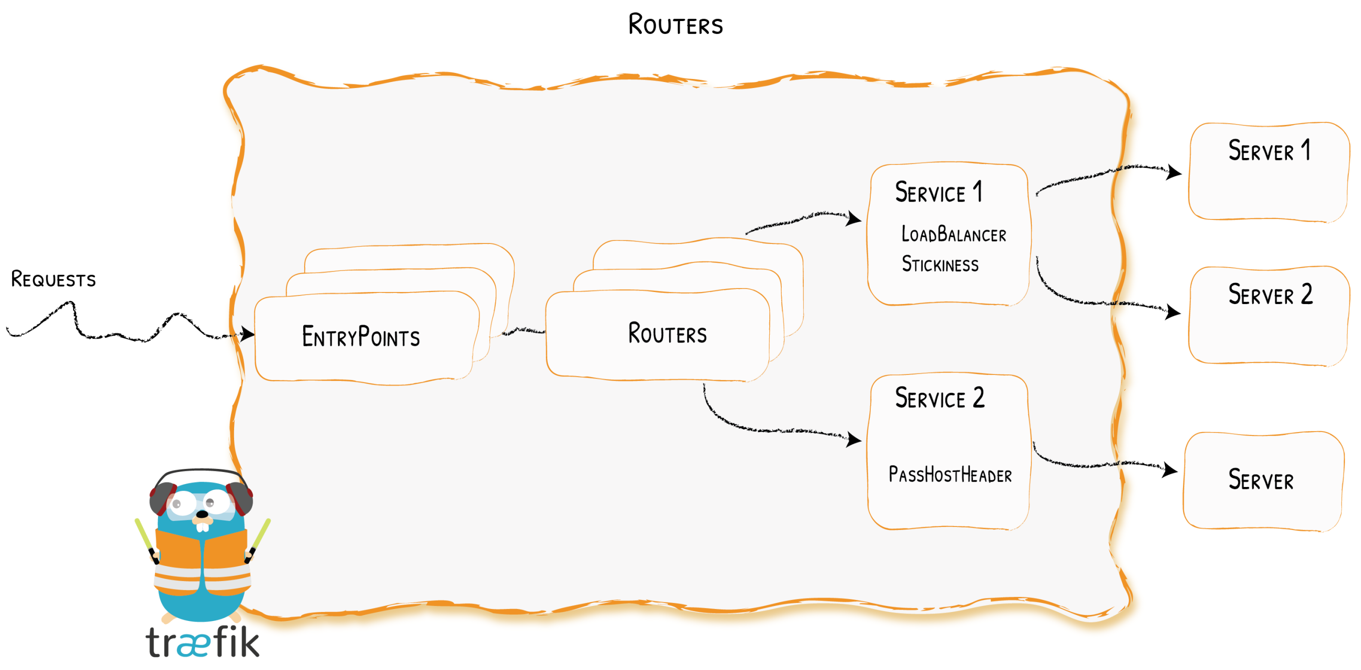


at next step it turns for routers. after that traffics received by entrypoints, it sends them to routers. in router based on rules that we set, if traffics match with those rules, we can decide to send that

traffic to service section or send it to middleware for more processing. picture below shows structure of routers.

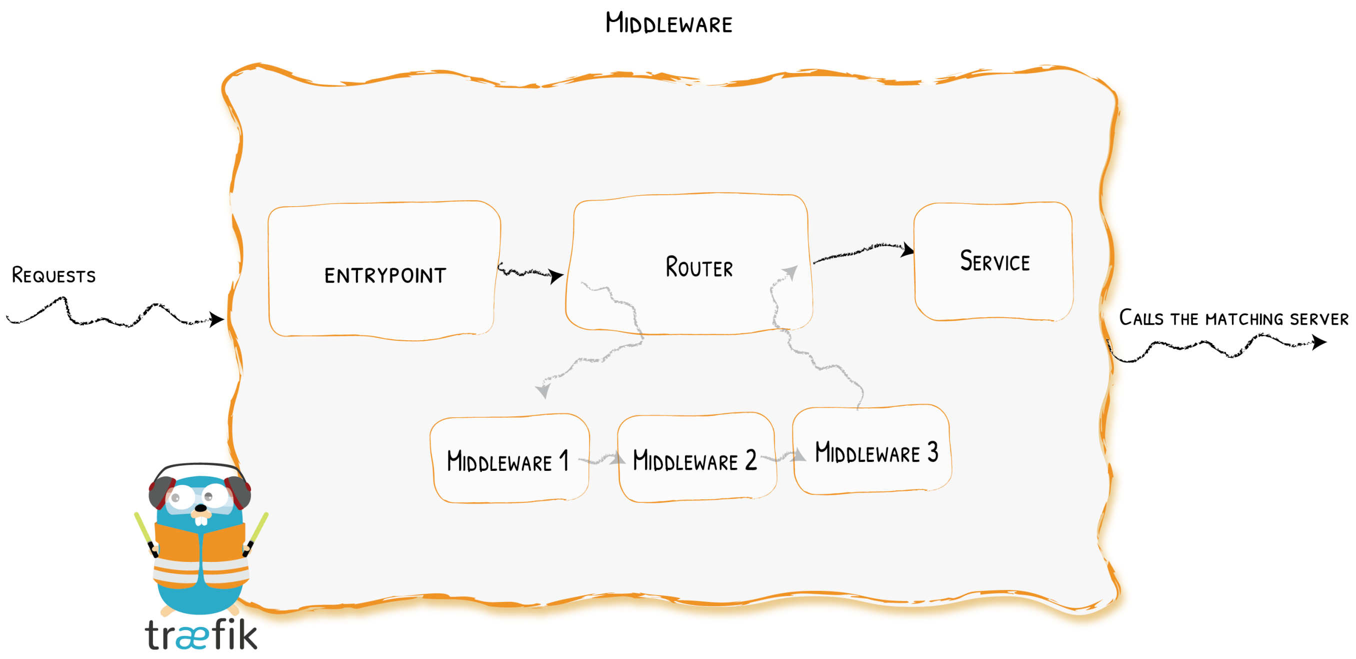


after that all conditions was ok, traffics have been sent to last step which is service. this part, points to our real application and access it by service name. service structure is like below.



the services are responsible for configuring how to reach the actual services that will eventually handle the incoming requests.

attached to the routers, pieces of middleware are a means of tweaking the requests before they are sent to our service. there are several available middleware in traefik, some can modify the request, the headers, some are in charge of redirections, some add authentication, and so on. from below picture we can find some basic structure of middlewares.



now we are going to install traefik instance on our kubernetes cluster using helm charts and configuring some modifications and authentications on our requests with middlewares.

for this installation we use helm charts of traefik. so we can follow steps below for continuing this guide.

```
step 1 : install traefik using helm charts:
```

```
git clone https://github.com/traefik/traefik-helm-chart.git
```

```
step 2 : configure values file. below codes, shows the changes we applied on values.yml file:
```

```
deployment:
  enabled: true
  kind: DaemonSet

ingressClass:
  enabled: true
  isDefaultClass: true

# For adding our static traefik config we can add to below section:
# additionalArguments:
#   - "--providers.kubernetesingress.ingressclass=traefik-internal"
#   - "--log-level=DEBUG"

# below section is entrypoints configuration.

ports:
  web:
    port: 8000
    expose: true
    exposedPort: 80
    protocol: TCP
    nodePort: 32080  ## we use nodePort type in traefik and load balance our requests with external HAproxy to 
this nodePort.
  websecure:
    port: 8443
    expose: true
    exposedPort: 443
    protocol: TCP
    nodePort: 32443

service:
  enabled: true
  type: NodePort
  spec:
    externalTrafficPolicy: Local  ## default is set to Cluster but to enable X-forward-for option we set this 
option to Local
```

step 3 : install traefik in default namespace (cd to values.yml directory):

```
helm install traefik .
```

step 4 : startup a nginx test deployment on kubernetes for testing purpose(nginx-test.yml).

step 5 : configure ingressroute workload to route our desired domain to nginx service(nginx-ingressroute.yml):

```
apiVersion: traefik.containo.us/v1alpha1
kind: IngressRoute
metadata:
  name: authz-ing
  namespace: apps
spec:
  entryPoints:
    - web
  routes:
    - match: Host(`authz.testcluster.local`) && Method(`POST`)
      kind: Rule
      middlewares:
        - name: whitelist
      services:
        - name: authz
          port: 80
    - match: Host(`authz.testcluster.local`) && Method(`GET`)
      kind: Rule
      services:
        - name: authz
          port: 80
```

in above workload I set the method condition for the domain. if method is "GET" it can access to service but if it is "POST" first it passes to a
middleware for checking the IP address of client. if IP of the client is in the list it can access to service using "POST" if not, the client
receives forbidden message.

step 6 : now we have to configure middleware workload(whitelist.yml).

```
apiVersion: traefik.containo.us/v1alpha1
kind: Middleware
metadata:
  name: whitelist
  namespace: apps
spec:
  ipWhiteList:
    sourceRange:
      - 89.37.12.158/32
```

in above workload we set an IP address to whitelist and if client address match with this address it can access to our domain using "POST" method.

step 7 : it's time to launch our system.

```
kubectl apply -f /opt/apps/whitelist.yml
kubectl apply -f /opt/apps/nginx-ingressroute.yml
kubectl apply -f /opt/apps/nginx-test.yml
```

step 8 : now we can test our application using curl to the domain specified in workload.