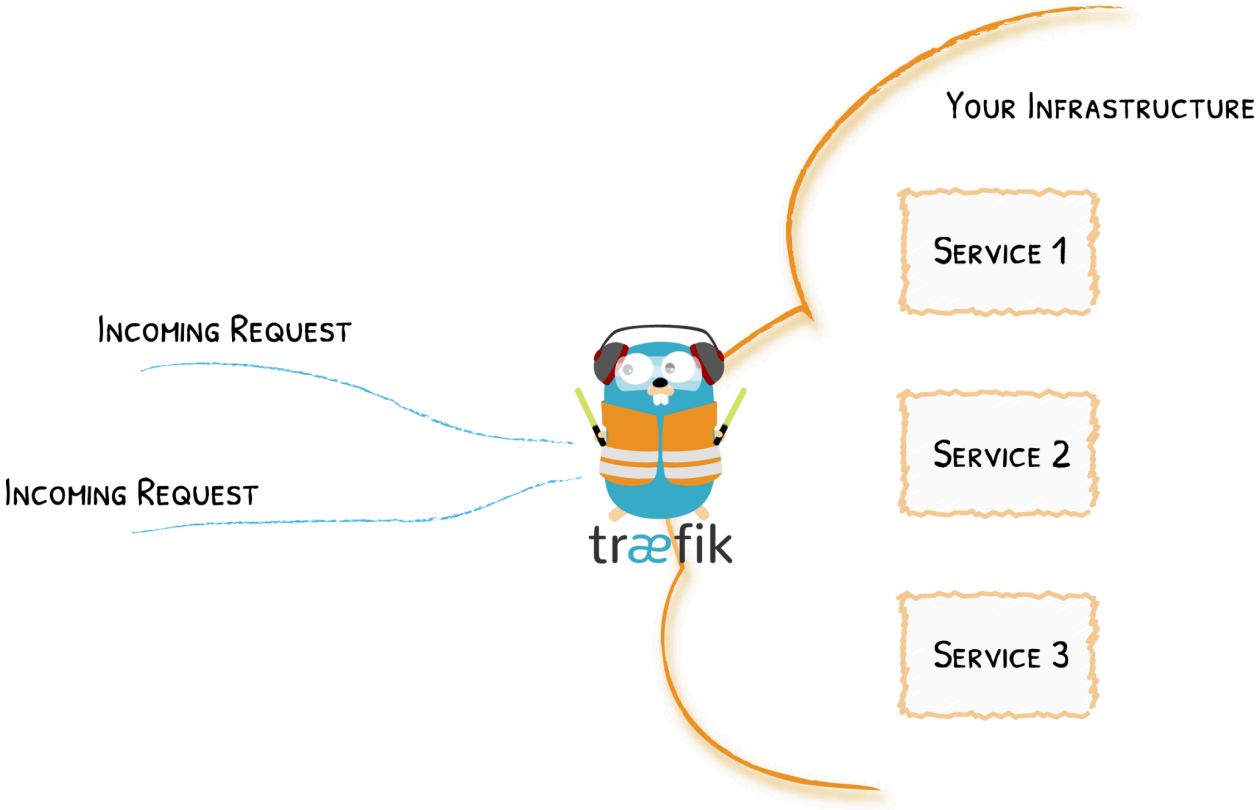
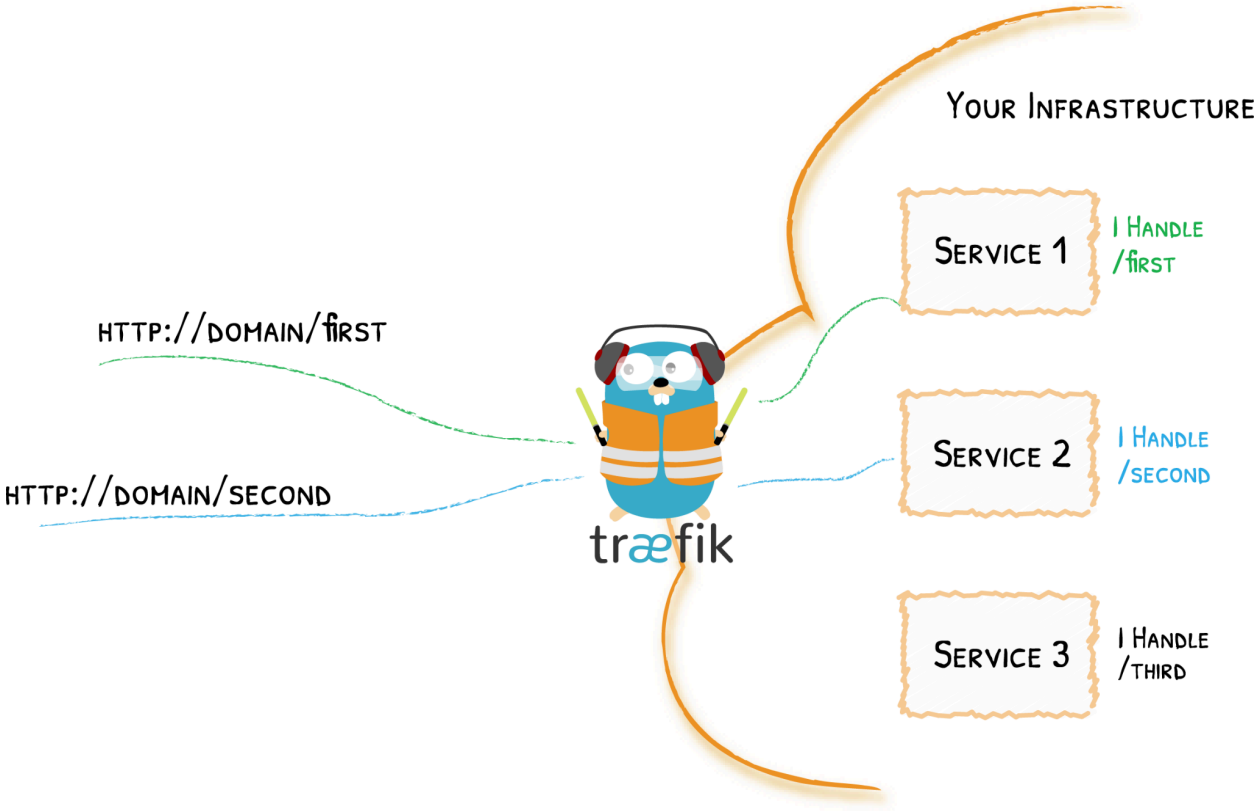
- Traefik operates on the principles of EntryPoints, Routers, Middlewares, and Services.
- EntryPoints serve as the entry points into Traefik's network. They specify the ports to receive packets and whether to listen for TCP or UDP traffic.
- Routers direct incoming requests to the appropriate services capable of handling them.
- Middlewares, linked to routers, have the ability to alter requests or responses before they reach the intended service.
- Services dictate the configuration for reaching the actual services responsible for processing incoming requests.
- Key features of Traefik include dynamic configuration, automatic service discovery, and support for various backends and protocols
-
- Traefik is an Edge Router; this means that it's the door to your platform, and that it intercepts and routes every incoming request: it knows all the logic and every rule that determines which services handle which requests (based on the path, the host, headers, etc.).



Traefik for Kubernetes:

Auto service discovery:

We attach information that tells Traefik the characteristics of the requests the services can handle. Whenever a service is deployed, Traefik detects it immediately and updates the routing rules in real time. Similarly, when a service is removed from the infrastructure, the corresponding route is deleted accordingly.



Traefik is able to use the cluster API to discover the services and read the attached information. In Traefik, these connectors are called providers because they provide the configuration to Traefik.

Traefik utilizes the Kubernetes API to discover services. For this, it requires permissions, managed through roles set by the cluster admin. First, a ClusterRole specifying permitted resources and actions is created. Then, it's bound to an account used by Traefik Proxy. This process ensures Traefik accesses Kubernetes API securely and effectively.

1-Role.yml

```yaml
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
 name: traefik-role
rules:
 - apiGroups:
 - ""
 resources:
 - services
 - endpoints
 - secrets
 verbs:
 - get
 - list
 - watch
 - apiGroups:
 - extensions
 - networking.k8s.io
 resources:
 - ingresses
 - ingressclasses
 verbs:
 - get
 - list
 - watch
 - apiGroups:
 - extensions
 - networking.k8s.io
 resources:
 - ingresses/status
 verbs:
 - update
```

2-Service-account.yml

```
apiVersion: v1
Kind: ServiceAccount
metadata:
  name: traefik-account
```

3-Role-binding.yml

```
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: traefik-role-binding
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: traefik-role
subjects:
  - kind: ServiceAccount
    name: traefik-account
    namespace: default
```

Install Traefik using Helm charts: (Kubernetes 1.16, +Helm version 3.9+)

```
helm repo add traefik https://traefik.github.io/charts

helm repo update

helm install --namespace=traefik-v2 traefik traefik/traefik
```

Two ways to expose the dashboard:

1. `kubectl port-forward $(kubectl get pods --selector "app.kubernetes.io/name=traefik" --output=name) 9000:9000`

Access at http://127.0.0.1:9000/dashboard/

2. **IngressRoute CRD:**

```
apiVersion: traefik.io/v1alpha1
```

```
kind: IngressRoute
metadata:
  name: dashboard
spec:
  entryPoints:
    - web
  routes:
    - match: Host(`traefik.localhost`) &&
(PathPrefix(`/dashboard`) || PathPrefix(`/api`))
      kind: Rule
      services:
        - name: api@internal
          kind: TraefikService
```

We can also deploy Traefik as DaemonSet/Deployment and service rather than using the Helm chart:

Traefik-deployment.yml

```
kind: Deployment
apiVersion: apps/v1
metadata:
  name: traefik-deployment
  labels:
    app: traefik
spec:
  replicas: 1
  selector:
    matchLabels:
      app: traefik
  template:
    metadata:
      labels:
        app: traefik
    spec:
      serviceAccountName: traefik-account
      containers:
        - name: traefik
          image: traefik:v2.11
          args:
            - --api.insecure
```

```
            - --providers.kubernetesingress
          ports:
            - name: web
              containerPort: 80
            - name: dashboard
              containerPort: 8080
```

DaemonSet:

```
kind: DaemonSet
apiVersion: apps/v1
metadata:
  name: traefik-daemonset
  labels:
    app: traefik
spec:
  selector:
    matchLabels:
      app: traefik
  template:
    metadata:
      labels:
        app: traefik
    spec:
      serviceAccountName: traefik-account
      containers:
        - name: traefik
          image: traefik:v2.11
          args:
            - --api.insecure
            - --providers.kubernetesingress
          ports:
            - name: web
              containerPort: 80
            - name: dashboard
```

Apply the configuration and create demo services:

3-whoami.yml

```yaml
kind: Deployment
apiVersion: apps/v1
metadata:
  name: whoami
  labels:
    app: whoami
spec:
  replicas: 3
  selector:
    matchLabels:
      app: whoami
  template:
    metadata:
      labels:
        app: whoami
    spec:
      containers:
        - name: whoami
          image: traefik/whoami
          ports:
            - name: web
              containerPort: 80
```

4-whoami-service.yml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: whoami
spec:
  ports:
    - name: web
      port: 80
      targetPort: web
  selector:
    app: whoami
```

Through integration with the Kubernetes API, Traefik receives notifications whenever an Ingress resource is created, updated, or deleted, enabling dynamic configuration. In this setup, I**ngress resources serve as the dynamic configuration for Traefik.**

5-ingress-resource.yml

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: whoami-ingress
spec:
  rules:
  - http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: whoami
            port:
              name: web
```

With host:

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: whoami-ingress
spec:
  rules:
  - host: yourhostname.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: whoami
            port:
              name: web
```

curl -v http://localhost/ to access the service which should now be loadbalancing

Traefik Configuration

Traefik's configuration consists of two main components:

1. **Static Configuration:**
    - This configuration sets up connections to providers and defines entry points where Traefik will listen for incoming traffic.
    - It includes settings that don't change frequently, such as network configurations and provider connections.
    - Static configuration elements establish the initial setup of Traefik and typically remain unchanged after deployment.
2. **Dynamic Configuration:**
    - The dynamic configuration defines how incoming requests are handled by the system.
    - It encompasses routing rules, middleware configurations, and service discovery settings.
    - Unlike static configuration, the dynamic configuration can change frequently and is hot-reloaded seamlessly by Traefik.
    - Changes to the dynamic configuration occur without interruption to requests or loss of connections, allowing for flexible and agile management of traffic handling.

The static configuration in Traefik can be defined using one of three methods, each mutually exclusive:

1. **Configuration File:**
    - **Traefik searches for static configuration in a file named `traefik.yml`, `traefik.yaml`, or `traefik.toml`.**
    - **The file can be located in various directories, such as `/etc/traefik/`, `$XDG_CONFIG_HOME/`, `$HOME/.config/`, or the working directory.**

- **You can specify a custom configuration file using the `--configFile` argument.**

2. **Command-line Arguments:**
    - **Static configuration options can also be provided via command-line arguments.**
    - **Use `traefik --help` or `docker run traefik[:version] --help` to view all available arguments.**
    - **Each argument corresponds to a specific configuration option.**
3. **Environment Variables:**
    - **Alternatively, static configuration options can be set as environment variables.**
    - **Refer to the static configuration environment overview for a list of available environment variables.**

If a value is not provided for an option, Traefik applies a default value. Sub-options also have default values if not specified. For instance, the `--providers.docker` option is sufficient to enable the Docker provider, even without specifying sub-options like `--providers.docker.endpoint`.

Traefik utilizes providers for configuration discovery, which are components that interface with various infrastructure systems such as orchestrators, container engines, cloud providers, or key-value stores. The purpose of providers is to retrieve relevant information about routing from these systems, and when changes occur, Traefik dynamically updates its routing configurations.

There are several types of providers, each belonging to one of four categories:

1. **Label-based Providers:** These providers rely on labels attached to deployed containers. Examples include Docker, Nomad, ECS, Marathon, and Rancher.

2. **Key-Value-based Providers:** These providers involve containers updating a key-value store with relevant information. Examples include Consul, Etcd, ZooKeeper, and Redis.
3. **Annotation-based Providers:** These providers utilize separate objects with annotations to define container characteristics. An example is Kubernetes Ingress.
4. **File-based Providers:** These providers use files to define configuration. Examples include file-based configuration and HTTP-based configuration.

Each provider operates within its namespace. For instance, if you declare a middleware using a Docker label, it resides in the Docker provider namespace. If you reference an object declared in another provider, the object name should be suffixed with `@<provider-name>.`

For Kubernetes, Traefik supports several providers:

- Docker: Orchestrator using labels.
- Kubernetes IngressRoute: Orchestrator using Kubernetes Custom Resource Definitions (CRDs).
- Kubernetes Ingress: Orchestrator using Kubernetes Ingress objects.
- Kubernetes Gateway API: Orchestrator using Kubernetes Gateway API Resource.

Additionally, Traefik provides configuration options to control the reload frequency of providers. The `providers.providersThrottleDuration` option specifies the duration Traefik waits after a configuration reload before considering new configuration changes from providers.

To restrict the scope of service discovery, Traefik offers two mechanisms:

1. **exposedByDefault:** A generic configuration option to control whether routes are created for all detected containers by default.

2. **Constraints:** A finer granularity mechanism based on constraints, allowing you to specify criteria for limiting route creation.

These mechanisms are supported by various providers such as Docker, ECS, Consul Catalog, Nomad, Rancher, Marathon, Kubernetes CRD, Kubernetes Ingress, and Kubernetes Gateway.