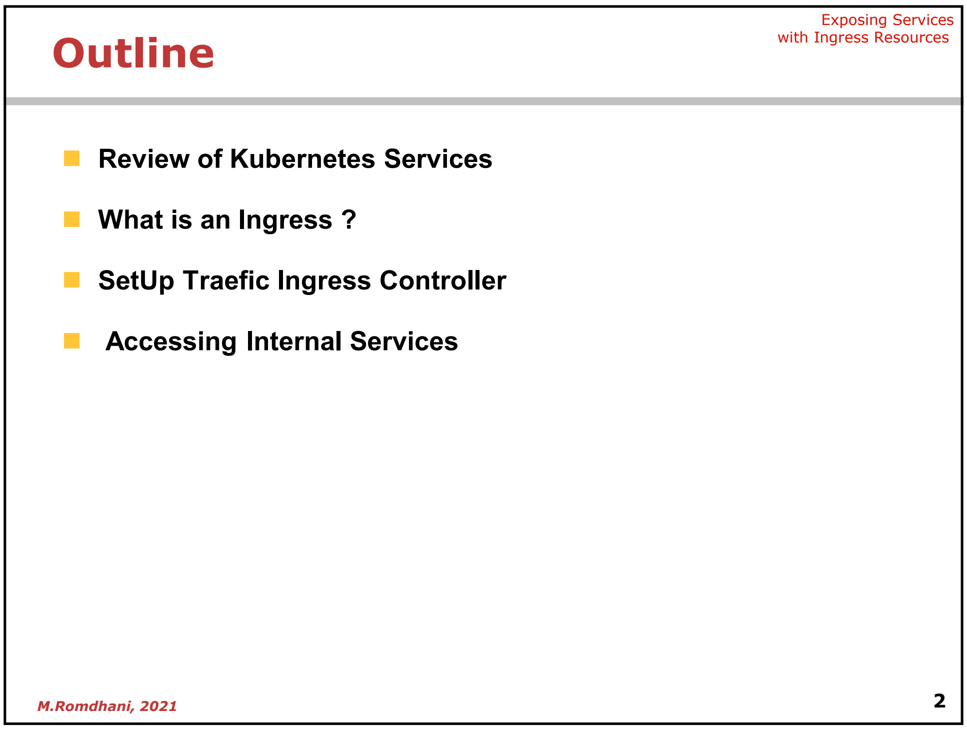
Unit 5

Exposing Services with Ingress Resources

Business Training

1



Outline

- Review of Kubernetes Services
- What is an Ingress ?
- SetUp Traefic Ingress Controller
- Accessing Internal Services

2

Review of Kubernetes Services

3

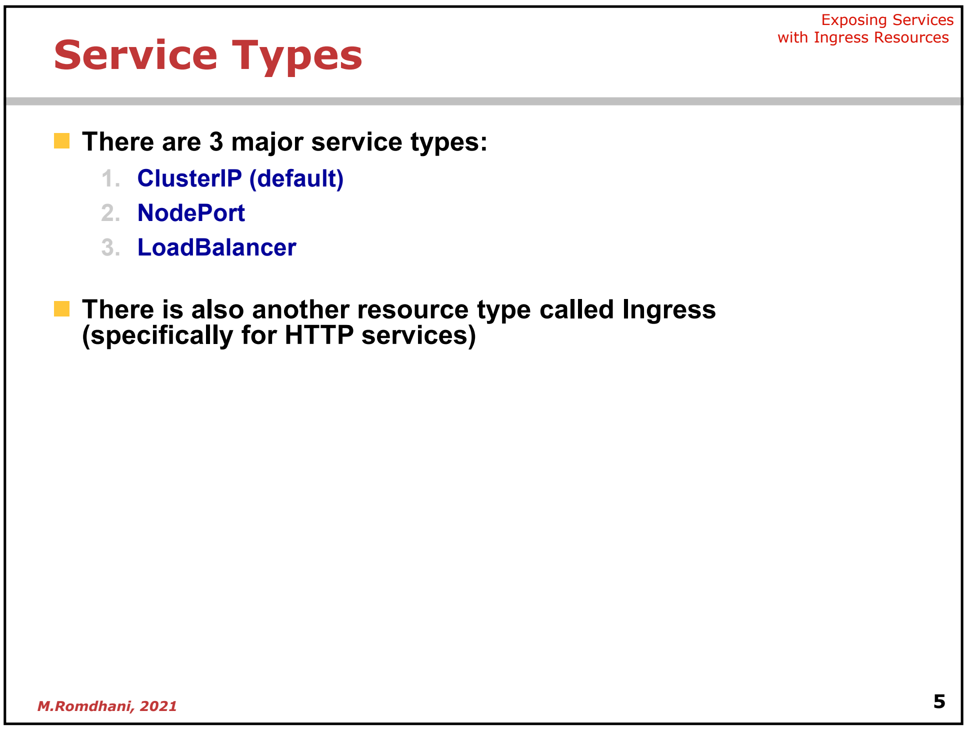
Services

- **Services give us a stable endpoint to connect to a pod or a group of pods**
  - Durable resource (unlike Pods)
    - static cluster-unique IP
  - Target Pods using equality based selectors
  - kube-proxy provides simple load-balancing.
- **An easy way to create a service is to use `kubectl expose`**
  - If we have a deployment named my-little-deploy, we can run:

```
kubectl expose deployment my-little-deploy --port=80
```

    - ... and this will create a service with the same name (`my-little-deploy`)
    - Services are automatically added to an internal DNS zone
    - In the example above, our code can now connect to `http://my-little-deploy/`
- **A service has a number of "endpoints"**

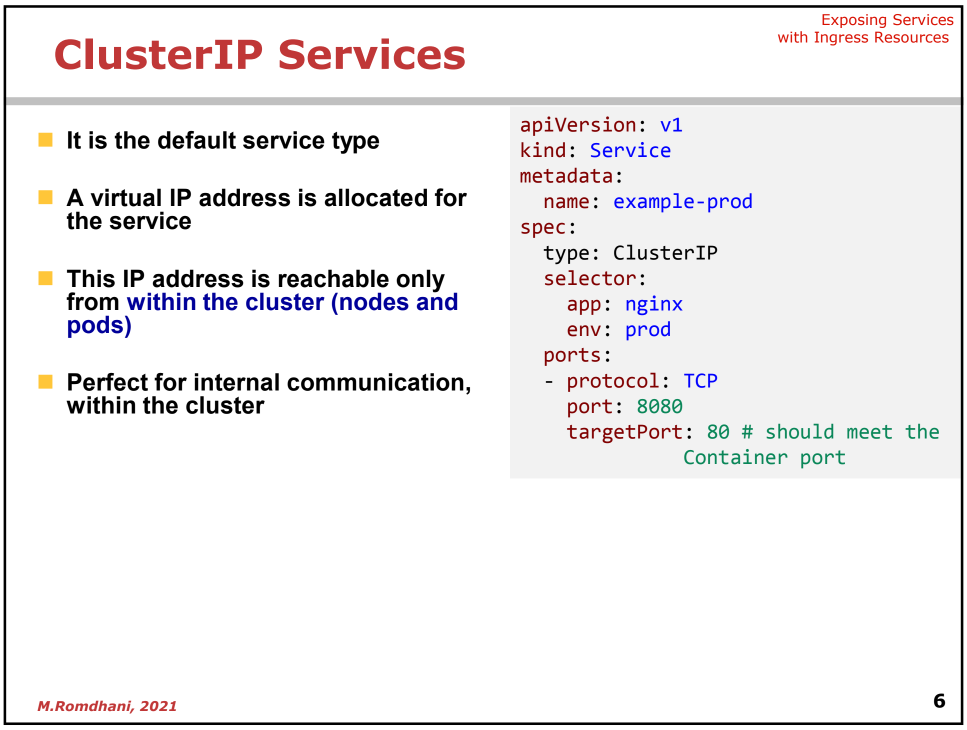
4



Service Types

- There are 3 major service types:
    1. ClusterIP (default)
    2. NodePort
    3. LoadBalancer

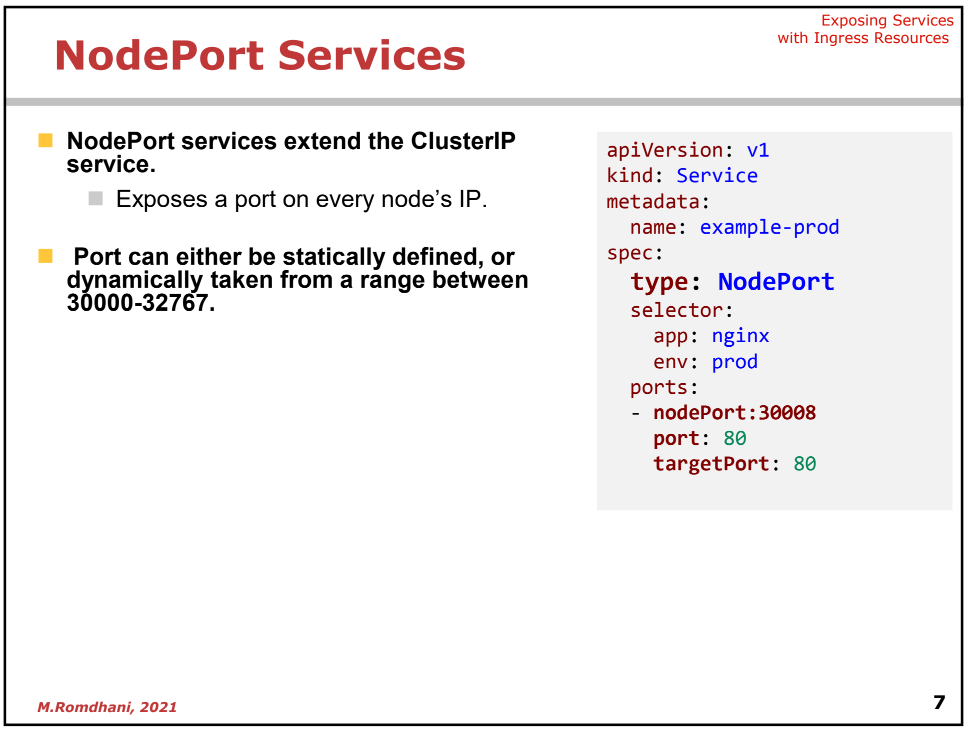
- There is also another resource type called Ingress (specifically for HTTP services)



ClusterIP Services

- It is the default service type
- A virtual IP address is allocated for the service
- This IP address is reachable only from within the cluster (nodes and pods)
- Perfect for internal communication, within the cluster

```
apiVersion: v1
kind: Service
metadata:
  name: example-prod
spec:
  type: ClusterIP
  selector:
    app: nginx
    env: prod
  ports:
  - protocol: TCP
    port: 8080
    targetPort: 80 # should meet the
              Container port
```

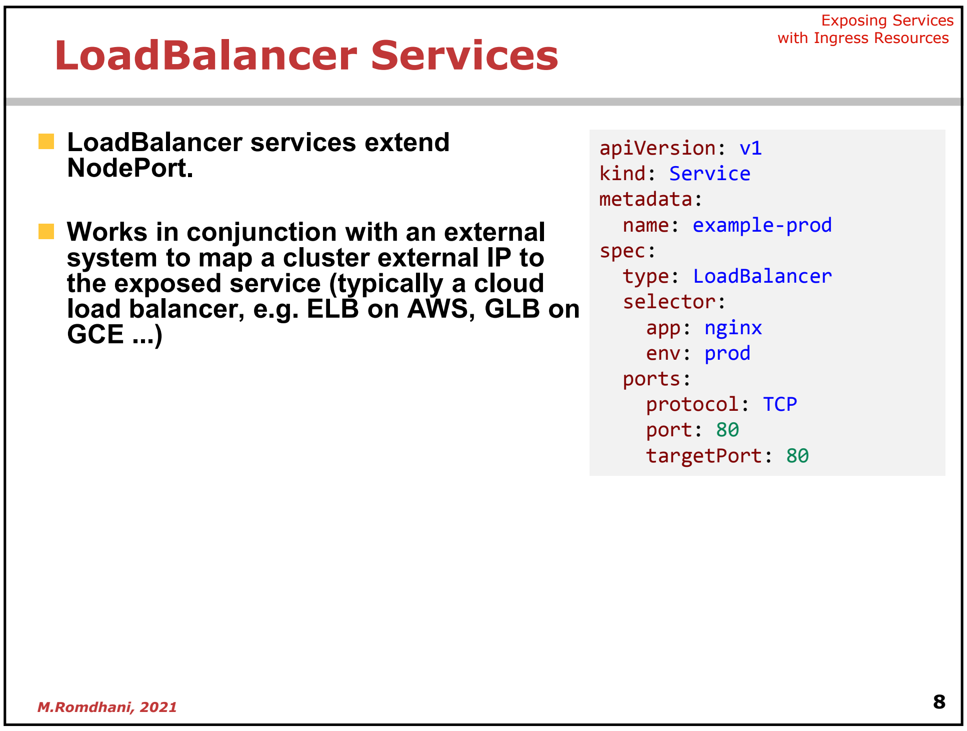


NodePort Services

- **NodePort services extend the ClusterIP service.**
  - Exposes a port on every node's IP.
- **Port can either be statically defined, or dynamically taken from a range between 30000-32767.**

```yaml
apiVersion: v1
kind: Service
metadata:
  name: example-prod
spec:
  type: NodePort
  selector:
    app: nginx
    env: prod
  ports:
  - nodePort:30008
    port: 80
    targetPort: 80
```

7



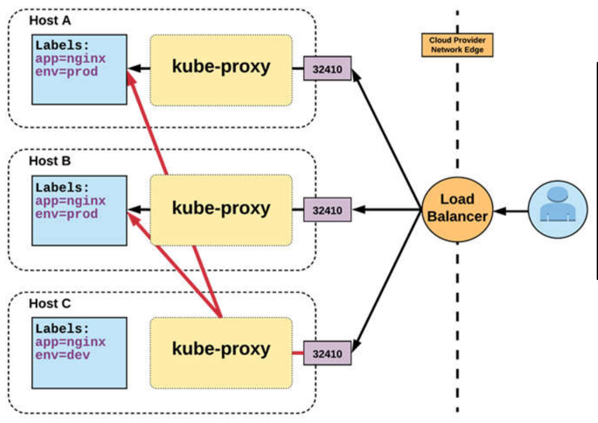
LoadBalancer Services

- **LoadBalancer services extend NodePort.**
- **Works in conjunction with an external system to map a cluster external IP to the exposed service (typically a cloud load balancer, e.g. ELB on AWS, GLB on GCE ...)**

```yaml
apiVersion: v1
kind: Service
metadata:
  name: example-prod
spec:
  type: LoadBalancer
  selector:
    app: nginx
    env: prod
  ports:
    protocol: TCP
    port: 80
    targetPort: 80
```

8

LoadBalancer Services



```
Name:        example-prod
Selector:    app=nginx,env=prod
Type:        LoadBalancer
IP:          10.96.28.176
LoadBalancer
Ingress:     172.17.18.43
Port:        <unset>  80/TCP
TargetPort:  80/TCP
NodePort:    <unset>  32410/TCP
Endpoints:   10.255.16.3:80,
             10.255.16.4:80
```

- Gets external IP from provider

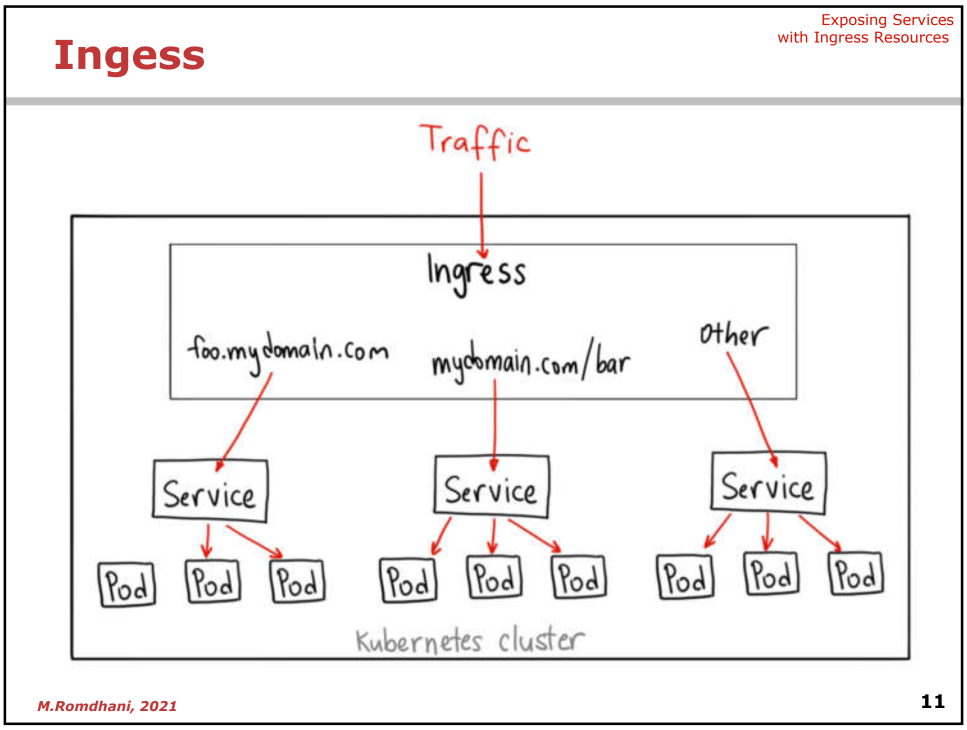
M.Romdhani, 2021

9

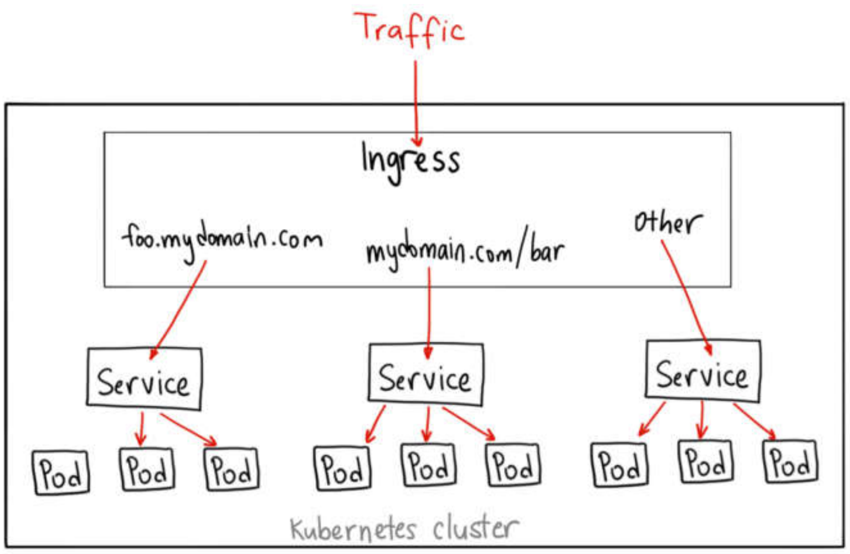
9

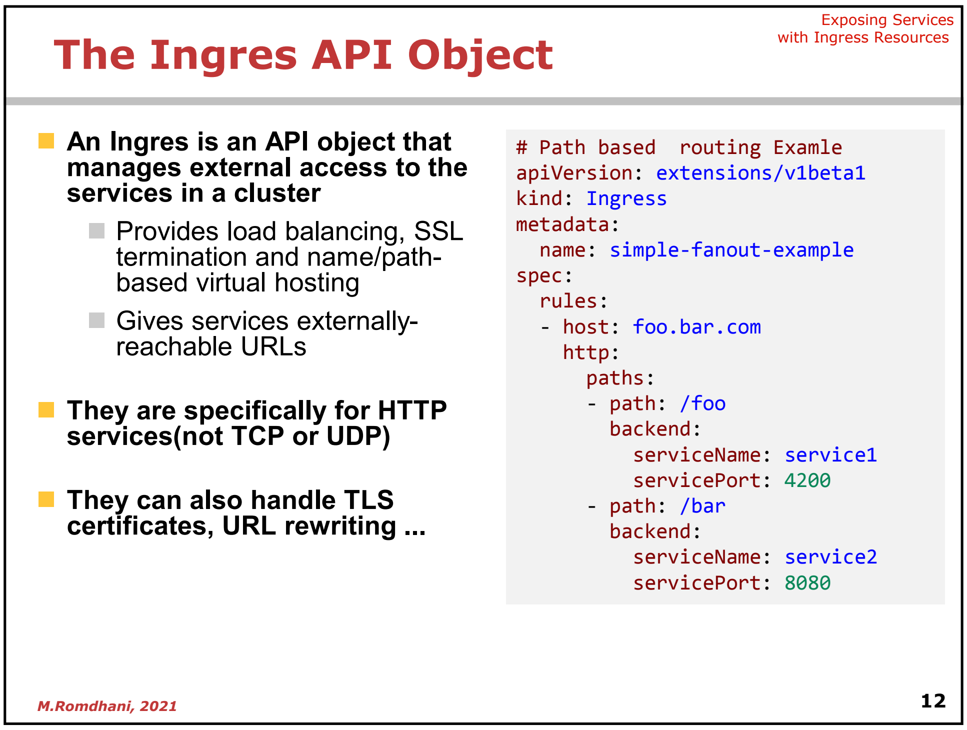
What is an Ingress ?

10



Ingess

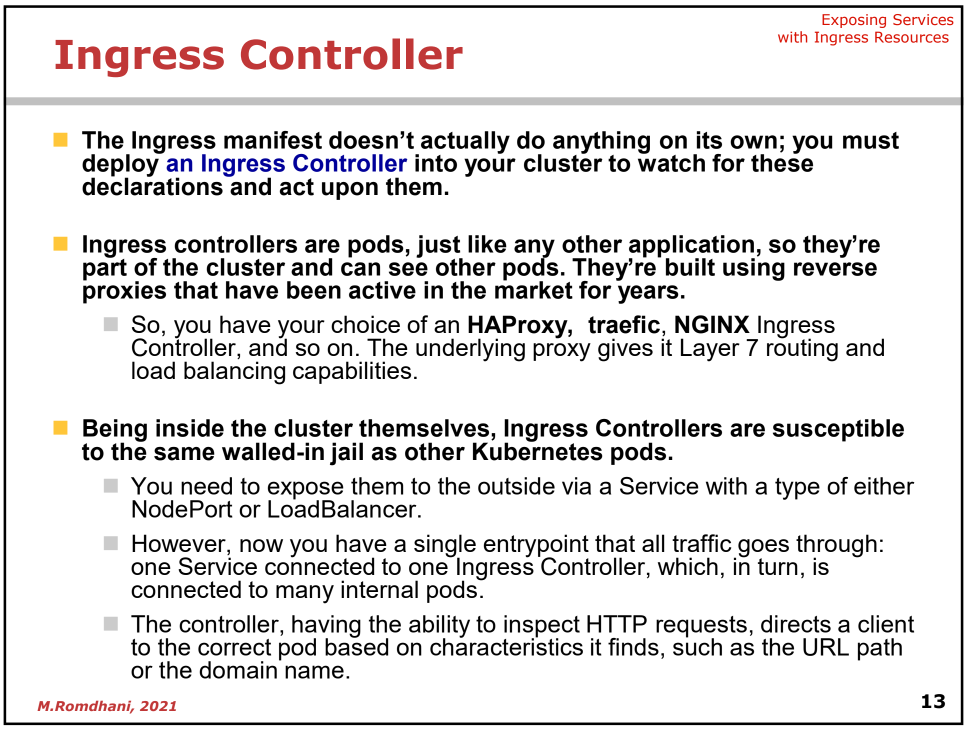




The Ingres API Object

- **An Ingres is an API object that manages external access to the services in a cluster**
  - Provides load balancing, SSL termination and name/path-based virtual hosting
  - Gives services externally-reachable URLs
- **They are specifically for HTTP services(not TCP or UDP)**
- **They can also handle TLS certificates, URL rewriting ...**

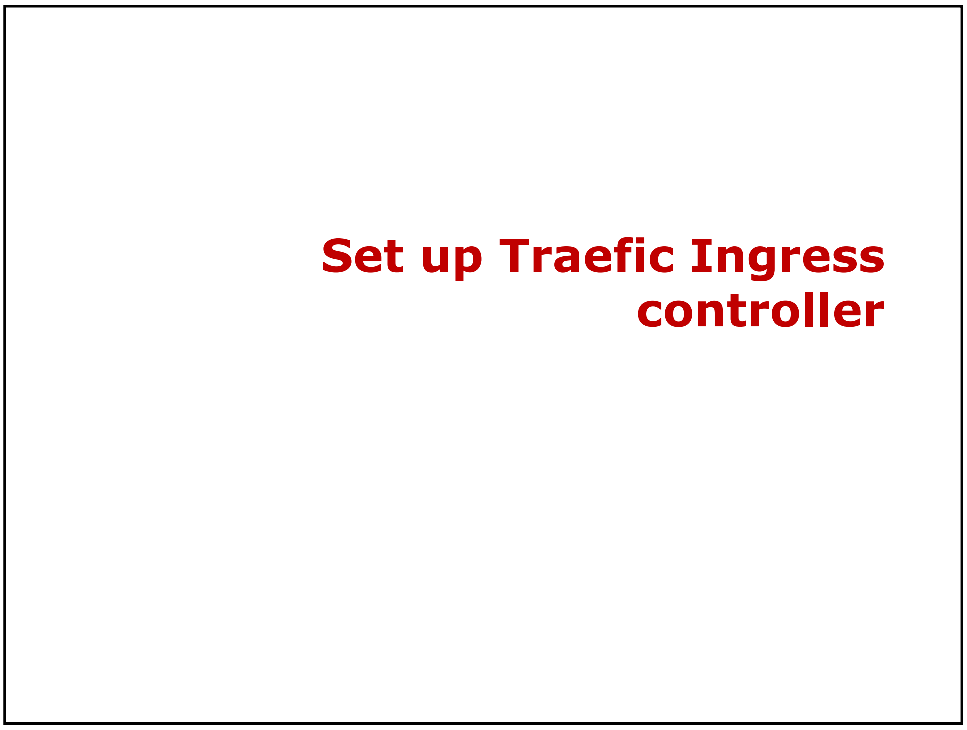
```
# Path based  routing Examle
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: simple-fanout-example
spec:
  rules:
  - host: foo.bar.com
    http:
      paths:
      - path: /foo
        backend:
          serviceName: service1
          servicePort: 4200
      - path: /bar
        backend:
          serviceName: service2
          servicePort: 8080
```



Ingress Controller

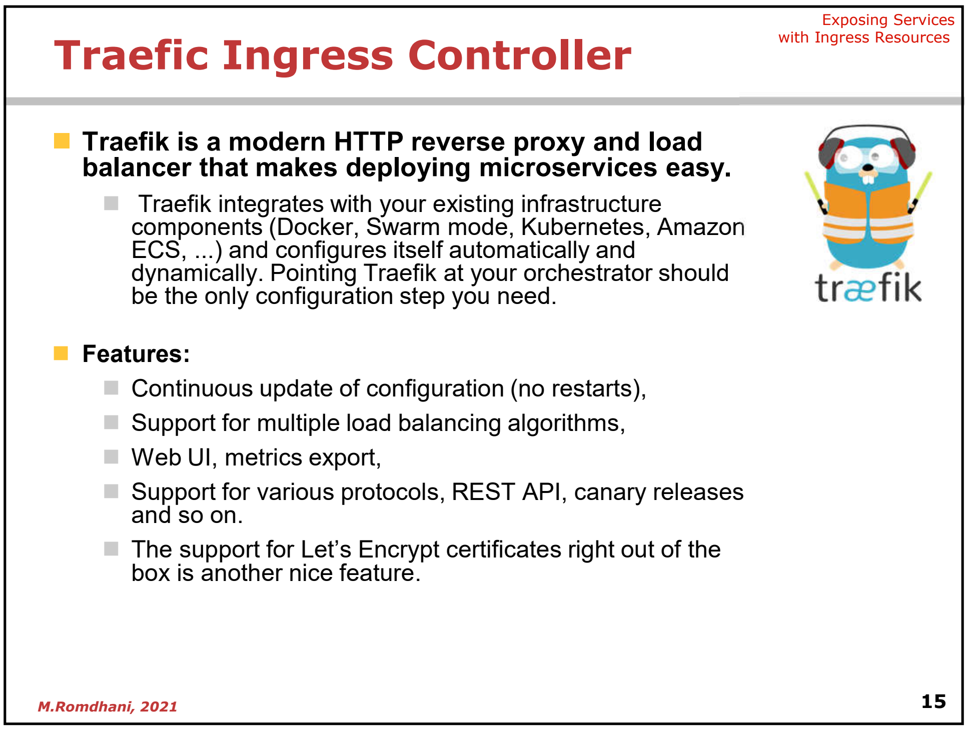
- **The Ingress manifest doesn't actually do anything on its own; you must deploy an Ingress Controller into your cluster to watch for these declarations and act upon them.**
- **Ingress controllers are pods, just like any other application, so they're part of the cluster and can see other pods. They're built using reverse proxies that have been active in the market for years.**
  - So, you have your choice of an **HAProxy**, **traefic**, **NGINX** Ingress Controller, and so on. The underlying proxy gives it Layer 7 routing and load balancing capabilities.
- **Being inside the cluster themselves, Ingress Controllers are susceptible to the same walled-in jail as other Kubernetes pods.**
  - You need to expose them to the outside via a Service with a type of either NodePort or LoadBalancer.
  - However, now you have a single entrypoint that all traffic goes through: one Service connected to one Ingress Controller, which, in turn, is connected to many internal pods.
  - The controller, having the ability to inspect HTTP requests, directs a client to the correct pod based on characteristics it finds, such as the URL path or the domain name.

13



Set up Traefic Ingress controller

14

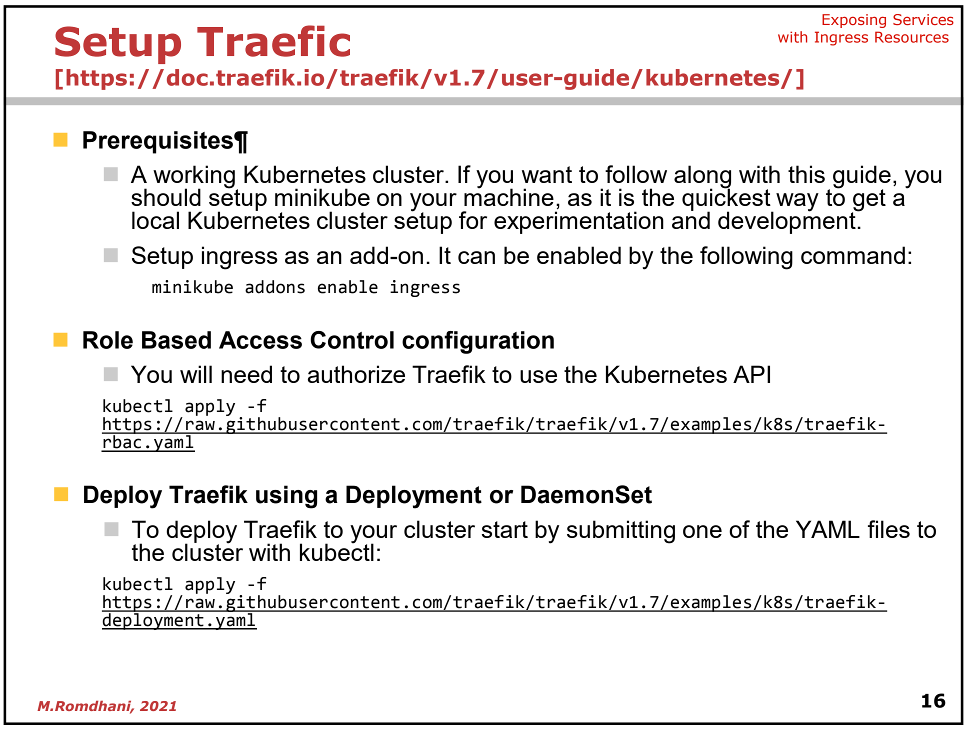


Traefic Ingress Controller

- **Traefik is a modern HTTP reverse proxy and load balancer that makes deploying microservices easy.**
  - Traefik integrates with your existing infrastructure components (Docker, Swarm mode, Kubernetes, Amazon ECS, ...) and configures itself automatically and dynamically. Pointing Traefik at your orchestrator should be the only configuration step you need.
- **Features:**
  - Continuous update of configuration (no restarts),
  - Support for multiple load balancing algorithms,
  - Web UI, metrics export,
  - Support for various protocols, REST API, canary releases and so on.
  - The support for Let's Encrypt certificates right out of the box is another nice feature.

M.Romdhani, 2021 **15**

15



Setup Traefic
[https://doc.traefik.io/traefik/v1.7/user-guide/kubernetes/]

- **Prerequisites¶**
  - A working Kubernetes cluster. If you want to follow along with this guide, you should setup minikube on your machine, as it is the quickest way to get a local Kubernetes cluster setup for experimentation and development.
  - Setup ingress as an add-on. It can be enabled by the following command:

```
minikube addons enable ingress
```

- **Role Based Access Control configuration**
  - You will need to authorize Traefik to use the Kubernetes API

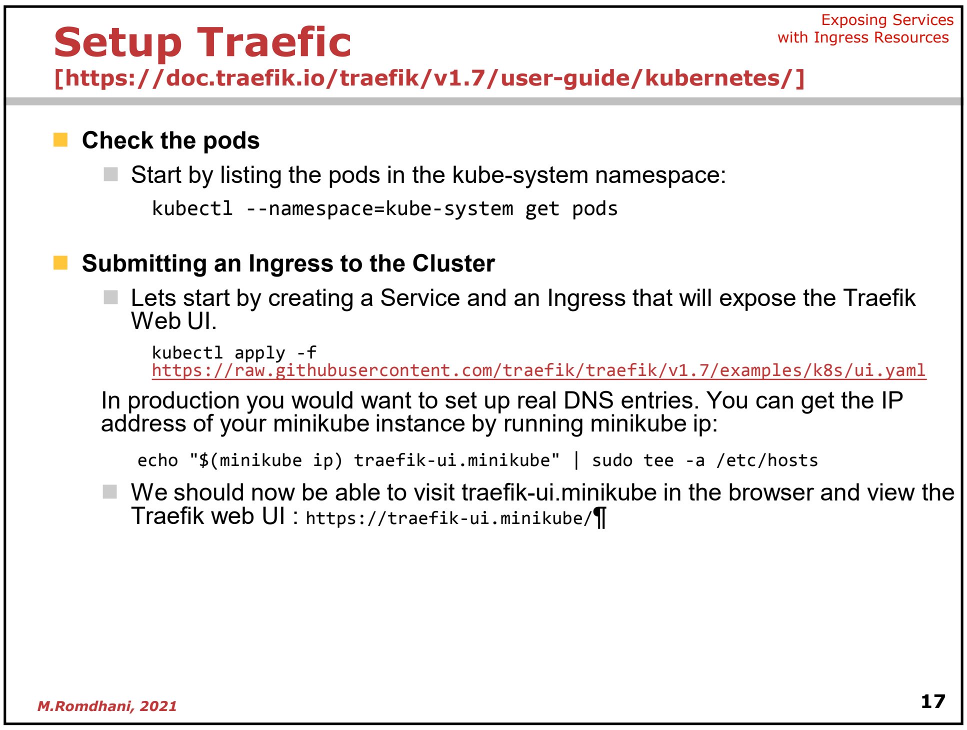
```
kubectl apply -f
https://raw.githubusercontent.com/traefik/traefik/v1.7/examples/k8s/traefik-
rbac.yaml
```

- **Deploy Traefik using a Deployment or DaemonSet**
  - To deploy Traefik to your cluster start by submitting one of the YAML files to the cluster with kubectl:

```
kubectl apply -f
https://raw.githubusercontent.com/traefik/traefik/v1.7/examples/k8s/traefik-
deployment.yaml
```

M.Romdhani, 2021 **16**

16



Setup Traefic
[https://doc.traefik.io/traefik/v1.7/user-guide/kubernetes/]

- **Check the pods**
  - Start by listing the pods in the kube-system namespace:

```
kubectl --namespace=kube-system get pods
```

- **Submitting an Ingress to the Cluster**
  - Lets start by creating a Service and an Ingress that will expose the Traefik Web UI.

```
kubectl apply -f
https://raw.githubusercontent.com/traefik/traefik/v1.7/examples/k8s/ui.yaml
```

In production you would want to set up real DNS entries. You can get the IP address of your minikube instance by running minikube ip:

```
echo "$(minikube ip) traefik-ui.minikube" | sudo tee -a /etc/hosts
```

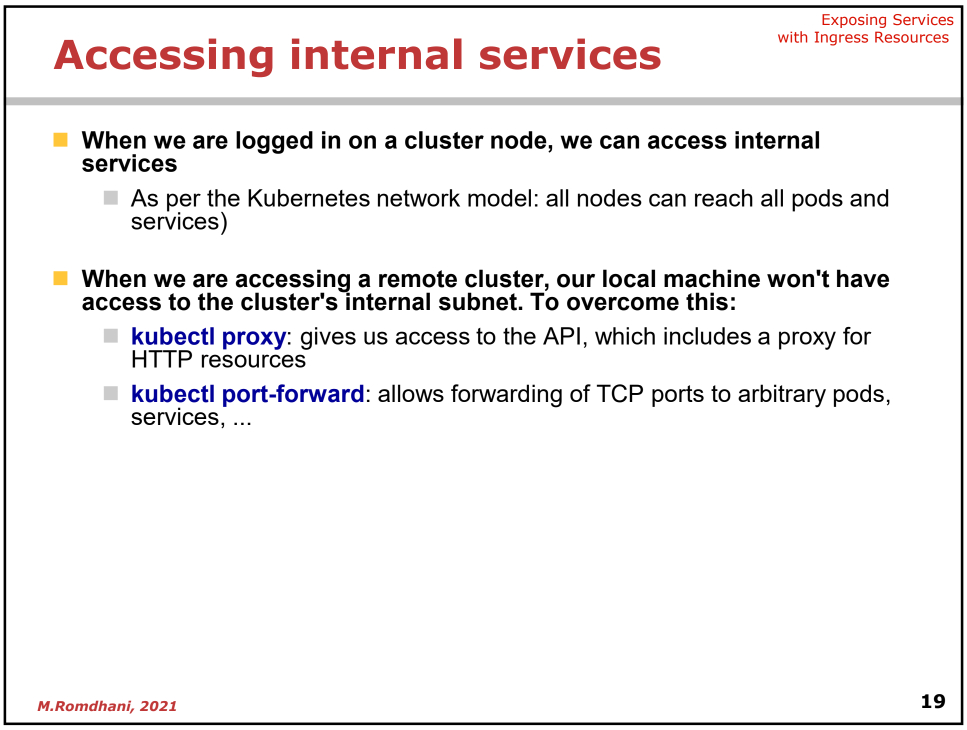
  - We should now be able to visit traefik-ui.minikube in the browser and view the Traefik web UI : `https://traefik-ui.minikube/`¶

M.Romdhani, 2021 **17**

17

Accessing Internal Services

18



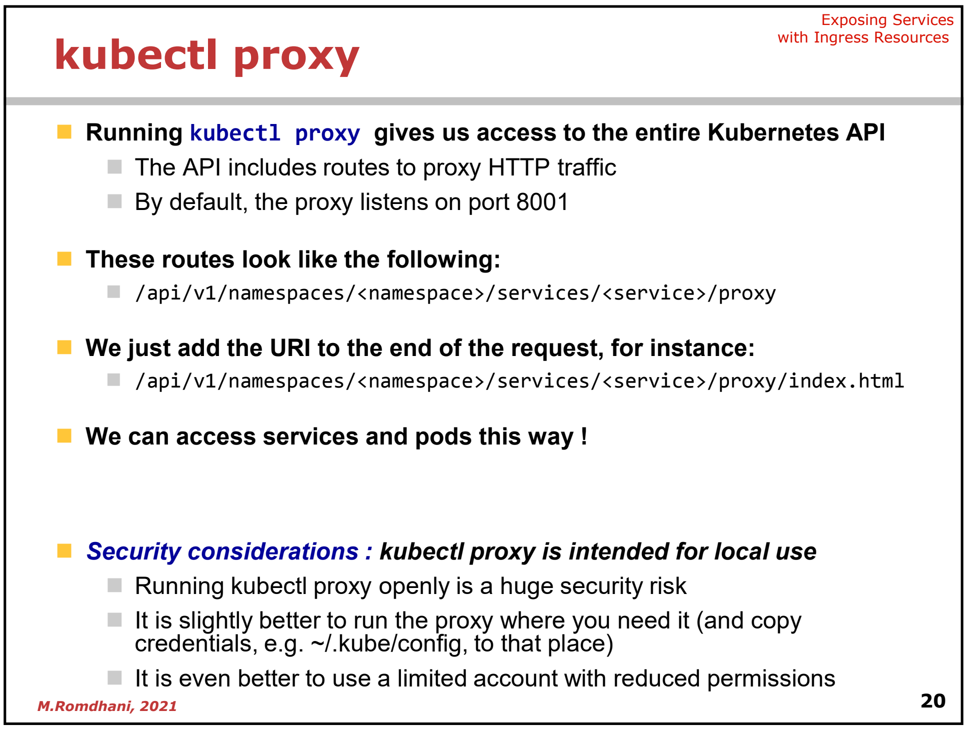
Exposing Services
with Ingress Resources

Accessing internal services

- **When we are logged in on a cluster node, we can access internal services**
  - As per the Kubernetes network model: all nodes can reach all pods and services)
- **When we are accessing a remote cluster, our local machine won't have access to the cluster's internal subnet. To overcome this:**
  - **kubectl proxy**: gives us access to the API, which includes a proxy for HTTP resources
  - **kubectl port-forward**: allows forwarding of TCP ports to arbitrary pods, services, ...

M.Romdhani, 2021 **19**

19



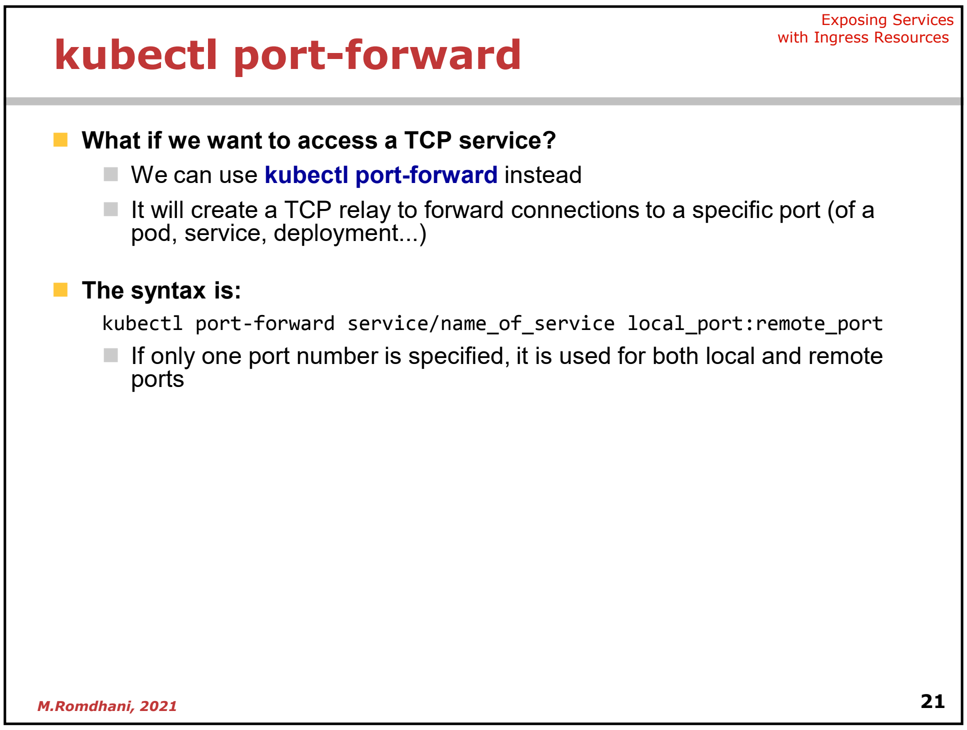
Exposing Services
with Ingress Resources

kubectl proxy

- **Running `kubectl proxy` gives us access to the entire Kubernetes API**
  - The API includes routes to proxy HTTP traffic
  - By default, the proxy listens on port 8001
- **These routes look like the following:**
  - `/api/v1/namespaces/<namespace>/services/<service>/proxy`
- **We just add the URI to the end of the request, for instance:**
  - `/api/v1/namespaces/<namespace>/services/<service>/proxy/index.html`
- **We can access services and pods this way !**
- ***Security considerations : kubectl proxy is intended for local use***
  - Running kubectl proxy openly is a huge security risk
  - It is slightly better to run the proxy where you need it (and copy credentials, e.g. ~/.kube/config, to that place)
  - It is even better to use a limited account with reduced permissions

M.Romdhani, 2021 **20**

20



kubectl port-forward

- **What if we want to access a TCP service?**
  - We can use **kubectl port-forward** instead
  - It will create a TCP relay to forward connections to a specific port (of a pod, service, deployment...)
- **The syntax is:**

```
kubectl port-forward service/name_of_service local_port:remote_port
```

  - If only one port number is specified, it is used for both local and remote ports