**Scenario (ingress)**

Let's use a hypothetical real-world e-commerce company as an example to illustrate how an Ingress resource can be beneficial.

\*\*E-Shop\*\*: A popular e-commerce platform.

**### Scenario:**

E-Shop has multiple microservices running in a Kubernetes cluster:

1. **\*\*Frontend Service\*\*:** Displays products, user profiles, and other UI elements.

2. **\*\*Products Service\*\*:** Manages product listings, details, and inventory.

3. **\*\*Cart Service\*\*:** Manages user carts, adding/removing products.

4. **\*\*Checkout Service\*\*:** Handles the checkout process, payments, and order confirmations.

**### How Ingress Helps:**

1. **\*\*Unified Entry Point\*\*:** Instead of exposing each service separately (which can be a security risk and management nightmare), E-Shop uses an Ingress to provide a single entry point to these services. This simplifies SSL/TLS management, logging, and monitoring.

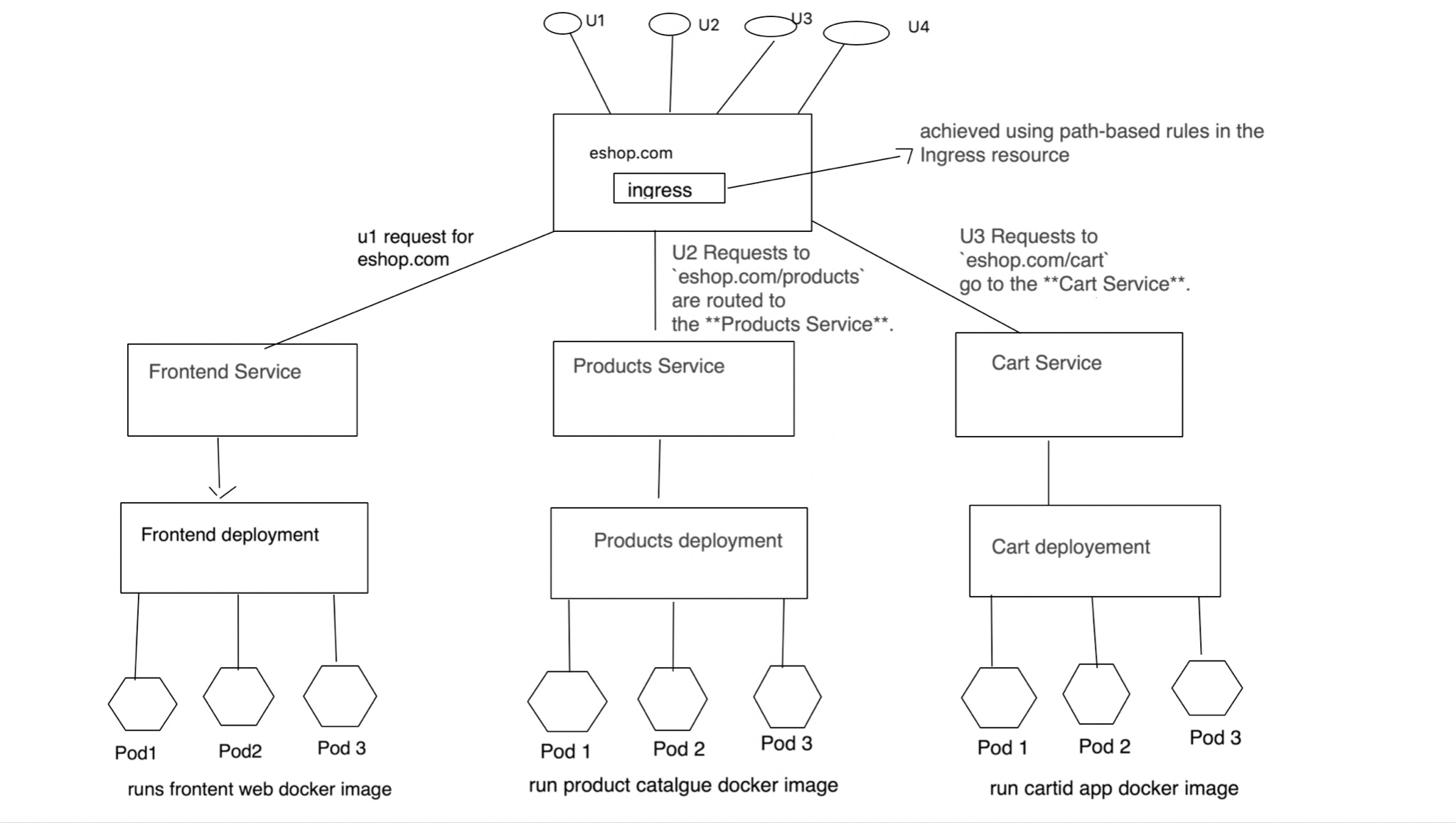
**2. \*\*Path-Based Routing\*\*:**

- Visitors accessing `eshop.com/` are directed to the \*\*Frontend Service\*\*.

- Requests to `eshop.com/products` are routed to the \*\*Products Service\*\*.

- Requests to `eshop.com/cart` go to the \*\*Cart Service\*\*.

- And, `eshop.com/checkout` directs users to the \*\*Checkout Service\*\*.



This is achieved using path-based rules in the Ingress resource, similar to the `/hello` path in the example you provided.

3. **\*\*Host-Based Routing (for multi-brand platforms)\***\*: If E-Shop owns multiple brands, they can route traffic based on domain names. For instance:

- `brandA.eshop.com` goes to Brand A's services.

- `brandB.eshop.com` goes to Brand B's services.

task 1 : create name space

>> Create a namespace by name : ing-internal

task 2 : create deployment

task 3 : create service

task 4 : create ingress controller

""Create a new nginx Ingress resource as follows:

✑ Name: pong

✑ Namespace: ing-internal

✑ Exposing service hello on path /hello using service port 5678""

**Task 1 : create namespace**

kubectl create ns ing-internal

**Task 2 : create a hello deployment**

vim hello-deployment.yaml

apiVersion: apps/v1

kind: Deployment

metadata:

name: hello

namespace: ing-internal

spec:

replicas: 2

selector:

matchLabels:

app: hello

template:

metadata:

labels:

app: hello

spec:

containers:

- name: hello-container

image: nginxdemos/hello:plain-text

ports:

- containerPort: 80

Deploy the hello app

kubectl create -f hello-deployment.yaml

The nginxdemos/hello:plain-text image is a simple image that serves a "Hello, World!" message over HTTP.

**Task 3 : Deploy the service**

This will create a service that exposes the hello deployment.

1. create yaml for service

vim hello-service.yaml

apiVersion: v1

kind: Service

metadata:

name: hello

namespace: ing-internal

spec:

selector:

app: hello

ports:

- protocol: TCP

port: 5678

targetPort: 80

**deploy the service**

kubectl create -f hello-service.yaml

**Notes : This service listens on nodeport 5678 and routes traffic to the hello pods on port 80.**

**task 4 : create ingress controller**

""Create a new nginx Ingress resource as follows:

✑ Name: pong

✑ Namespace: ing-internal

✑ Exposing service hello on path /hello using service port 5678""

1. create a yaml file

vim pong-ingress.yaml

#copy below contents to yaml file

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

name: pong

namespace: ing-internal

annotations:

nginx.ingress.kubernetes.io/rewrite-target: /

spec:

rules:

- http:

paths:

- path: /hello

pathType: Prefix

backend:

service:

name: hello

port:

number: 5678

let's break this down into simple terms:

* Imagine you have a building (your Kubernetes cluster). Inside this building, there's a room (namespace) called `ing-internal`. In this room, you have a door (the `Ingress` resource) named `pong`.
* Now, this door (`pong`) has some special instructions (annotations) on it. One of these instructions tells the door's security guard (nginx Ingress controller) that when someone enters asking for the `/hello` room, they should be directed straight inside without asking for the specific room name again (that's what the `nginx.ingress.kubernetes.io/rewrite-target: /` annotation does).
* The door has a list of rules about who can enter and where they should go. One of these rules says that if someone comes to the door asking for the `/hello` path, they should be directed to a specific desk (service) named `hello` inside the room. This desk is located at spot number `5678` (port 5678).
* In essence, when someone (or some data) comes to this building and goes to the `pong` door in the `ing-internal` room asking for `/hello`, they'll be directed to the `hello` desk at spot number `5678`.

**Here's a breakdown of the YAML:**

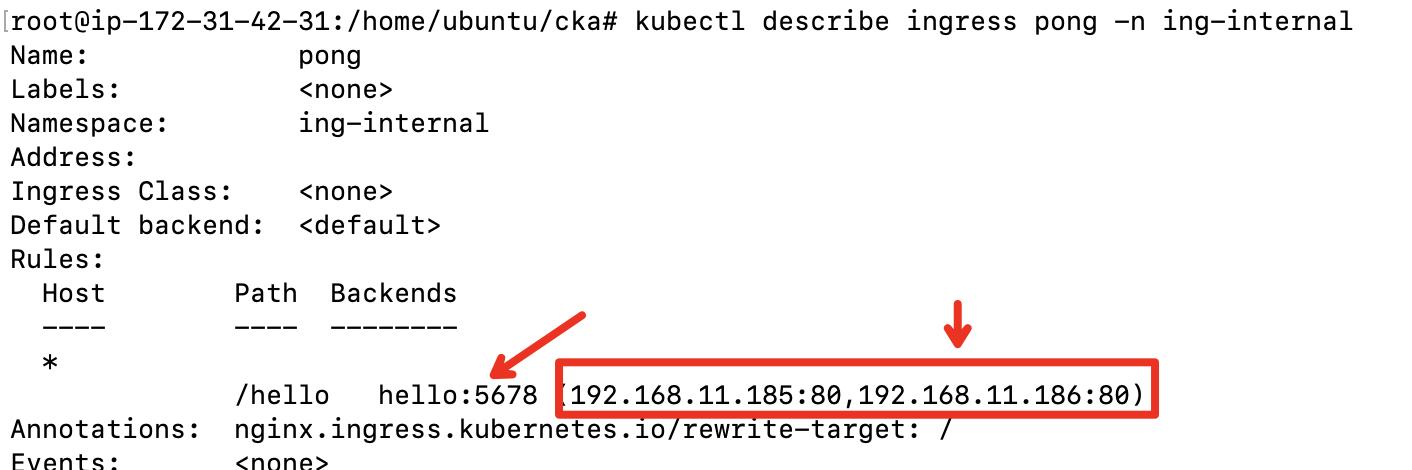
* apiVersion**: Specifies the API version to use. For Ingress resources,** networking.k8s.io/v1 **is the stable version as of Kubernetes 1.19+.**
* kind**: Specifies the kind of resource. In this case, it's** Ingress**.**
* metadata**: Contains metadata about the Ingress, such as its name and namespace.**
* annotations**: Contains any annotations for the Ingress. The** nginx.ingress.kubernetes.io/rewrite-target: / **annotation is used to rewrite the URL path to** / **when forwarding the request to the backend service. This is often used when the service doesn't need to know the original path.**
* spec**: Contains the specification for the Ingress.**
* rules**: Contains a list of rules for how to route traffic.**
* http**: Specifies that this rule is for HTTP traffic.**
* paths**: Contains a list of paths and their associated backends.**
* path**: Specifies the path to match. In this case, it's** /hello**.**
* pathType**: Specifies how to match the path.** Prefix **means any path that starts with** /hello **will match.**
* backend**: Specifies where to send the traffic that matches this path.**
* service**: Specifies the service to send the traffic to.**
* name**: The name of the service. In this case, it's** hello**.**
* port**: Specifies the port on the service to send the traffic to. In this case, it's** 5678**.**

1. create a yaml file

kubectl create -f pong-ingress.yaml

1. verify

kubectl describe ingress pong -n ing-internal

****

**You can notice 2 POD ips serving the request**

**================ lab Completion =====================**