

# Sequential Breakdown of the Process

In this lesson, we will first go through the sequential breakdown of Ingress resource creation process and then create the second Ingress resource.

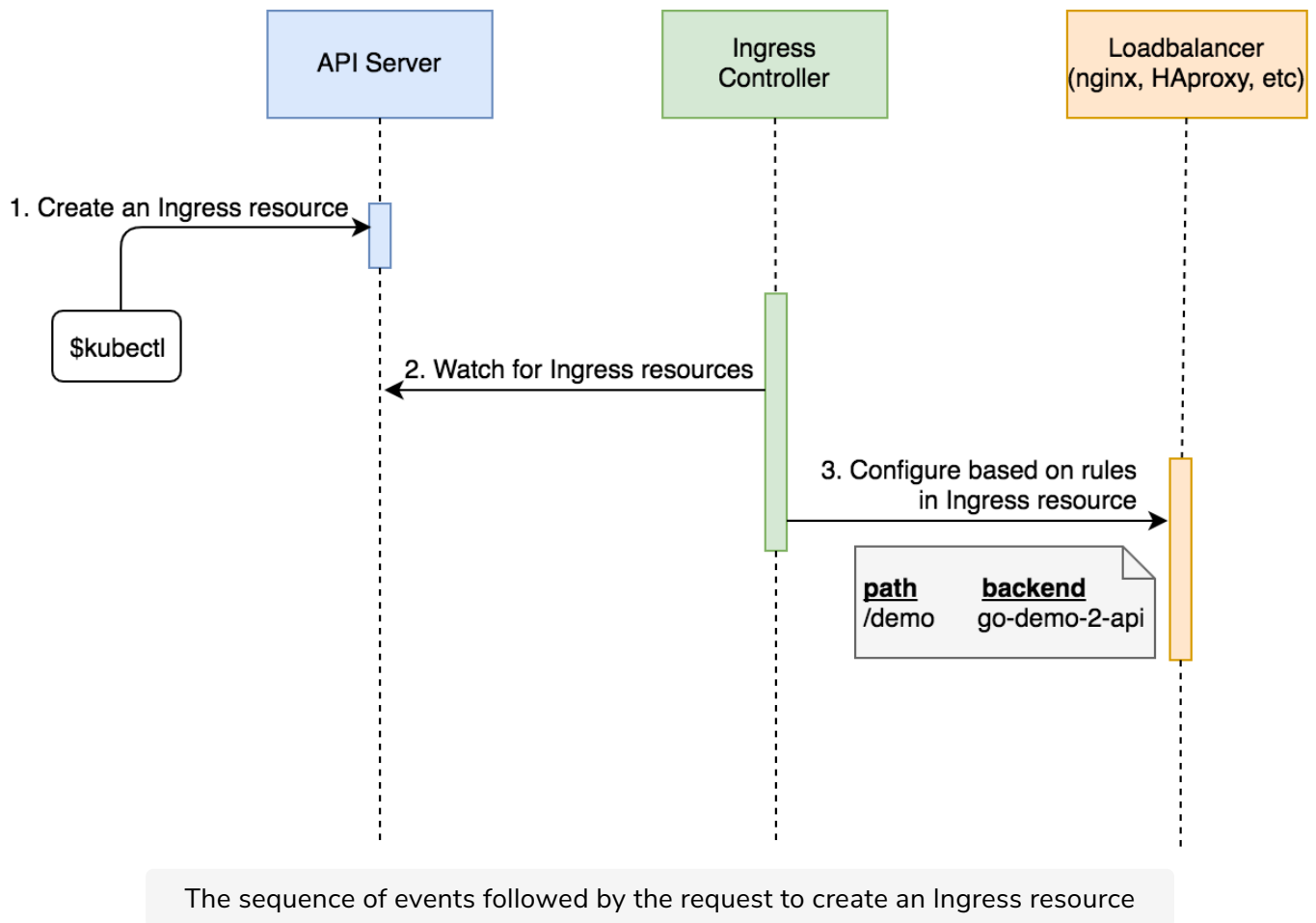
## WE'LL COVER THE FOLLOWING



- Creating Second Ingress Resource
  - Looking into the Definition
  - Deleting and Recreating the Objects

Let's see, through a sequence diagram, what happened when we created the Ingress resource.

1. The Kubernetes client ( `kubectl` ) sent a request to the API server requesting the creation of the Ingress resource defined in the `ingress/go-demo-2.yml` file.
2. The ingress controller is watching the API server for new events. It detected that there is a new Ingress resource.
3. The ingress controller configured the load balancer. In this case, it is `nginx` which was enabled by `minikube addons enable ingress` command. It modified `nginx.conf` with the values of all `go-demo-2-api` endpoints.



Now that one of the applications is accessible through Ingress, we should apply the same principles to the other.

## Creating Second Ingress Resource #

Let's first look into the definition and then create our second resource using `devops-toolkit.yml`.

### Looking into the Definition #

Let's take a look at the full definition of all the objects behind the `devops-toolkit` application.

```
cat ingress/devops-toolkit.yml
```

The **output**, limited to the Ingress object, is as follows.

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: devops-toolkit
```

```

annotations:
  kubernetes.io/ingress.class: "nginx"
  ingress.kubernetes.io/ssl-redirect: "false"

  nginx.ingress.kubernetes.io/ssl-redirect: "false"
spec:
  rules:
  - http:
      paths:
      - path: /
        backend:
          serviceName: devops-toolkit
          servicePort: 80
  ...

```

The `devops-toolkit` Ingress resource is very similar to `go-demo-2`.

The only significant difference is that the `path` is set to `/`.

It will serve all requests. It would be a much better solution if we'd change it to a unique base path (e.g., `/devops-toolkit`) since that would provide a unique identifier.

However, this application does not have an option to define a base path, so an attempt to do so in Ingress would result in a failure to retrieve resources. We'd need to write `rewrite` rules instead. We could, for example, create a rule that rewrites path base `/devops-toolkit` to `/`.

That way if, for example, someone sends a request to `/devops-toolkit/something`, Ingress would rewrite it to `/something` before sending it to the destination Service. While such an action is often useful, we'll ignore it for now. For now, `/` as the base `path` should do.

Apart from adding Ingress to the mix, the definition removed `type: NodePort` from the Service. This is the same type of action we did previously with the `go-demo-2` service. We do not need external access to the Service.

## Deleting and Recreating the Objects #

Let's create the objects defined in the `ingress/devops-toolkit.yml` file.

```

kubectl create \
  -f ingress/devops-toolkit.yml \
  --record --save-config

```



Let's take a look at the Ingresses running inside the cluster.

```
kubectl get ing
```



The **output** is as follows.

NAME	HOSTS	ADDRESS	PORTS	AGE
devops-toolkit	*	192.168.99.100	80	20s
go-demo-2	*	192.168.99.100	80	58s



We can see that now we have multiple Ingress resources. The Ingress Controller (in this case NGINX) configured itself taking both of those resources into account.

We can define multiple Ingress resources that will configure a single Ingress Controller.


Let's confirm that both applications are accessible through HTTP (port **80**).

```
open "http://$IP"
```

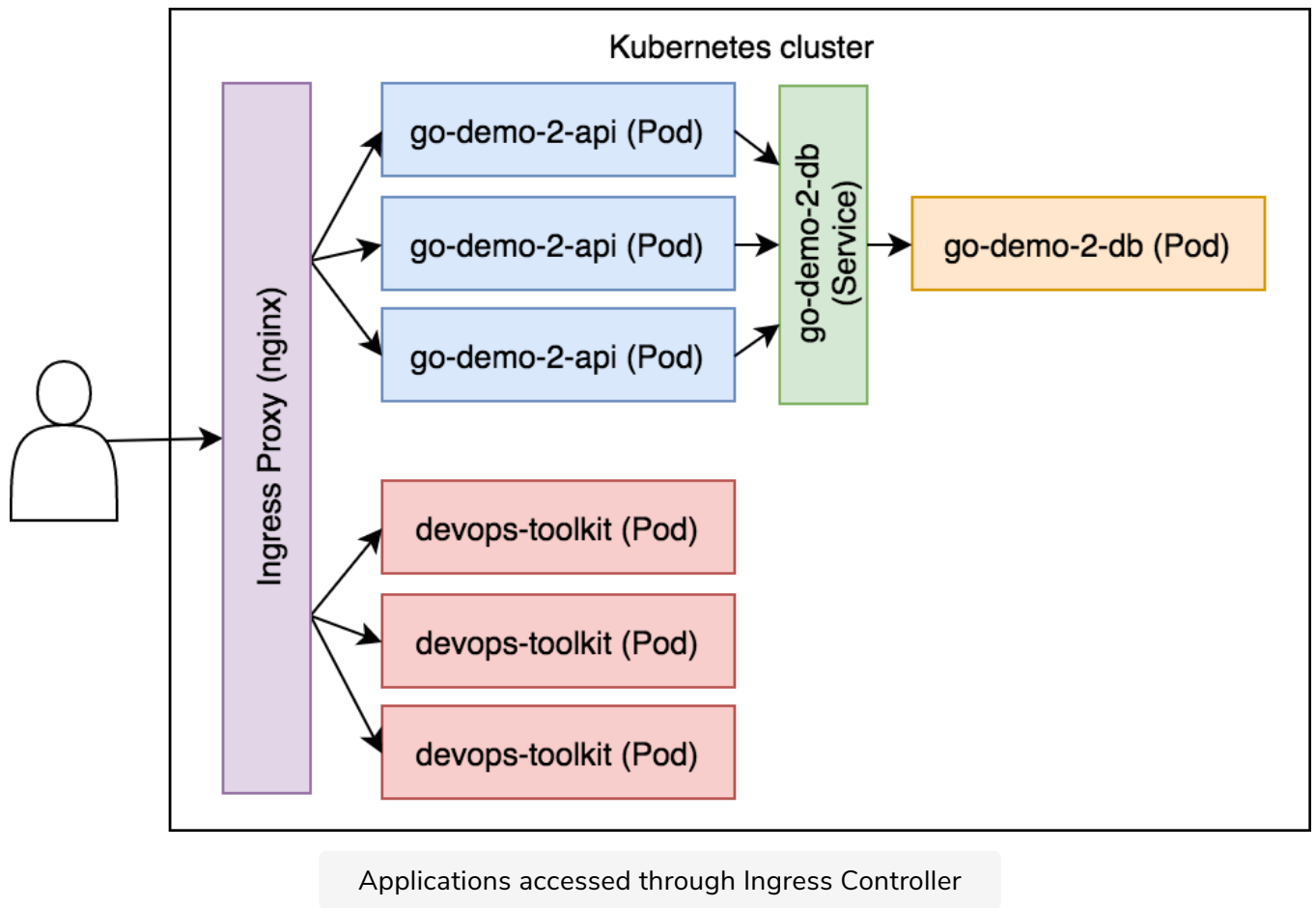
```
curl "http://$IP/demo/hello"
```



The first command opened one of the applications in a browser, while the other returned the already familiar **hello, world!** message.

 If you cannot connect, try running **IP=\$(minikube ip)** before the above commands.

Ingress is a (kind of) Service that runs on all nodes of a cluster. A user can send requests to any and, as long as they match one of the rules, they will be forwarded to the appropriate Service.



Even though we can send requests to both applications using the same port (80), that is often a sub-optimal solution. Our users would probably be happier if they could access those applications through different domains.

In the next lesson, we will explore creating Ingress resources based on domains.