Kubernetes Project: Exploring Ingress, Services, and Traffic Management

By: Aryan Gadhavi | DevOps Enthusiast

CONTENTS

Why Kubernetes?

1. **Scalability**
   Seamlessly scale applications for growing workloads and dynamic demands.
2. **Automation**
   Automate deployment, resource allocation, and failure recovery effortlessly.
3. **Resilience**
   Ensures high availability and fault-tolerance for reliable application performance.
4. **Traffic Management**
   Efficiently routes traffic and manages communication between services.
5. **DevOps Integration**
   Simplifies CI/CD workflows, enabling faster delivery and improved collaboration.

Project-Objectives

01 Traffic Routing

Automatically scales applications based on resource demands and traffic load.

02 Service Exposure

Ensures service reliability through self-healing, failover, and load balancing.

03 Configuration Simplicity

Leverage YAML configurations for streamlined deployment and management.

04 Scalability

Enable dynamic scaling to handle workload changes effectively.

Ingress Features

1. HTTP and HTTPS Traffic Management

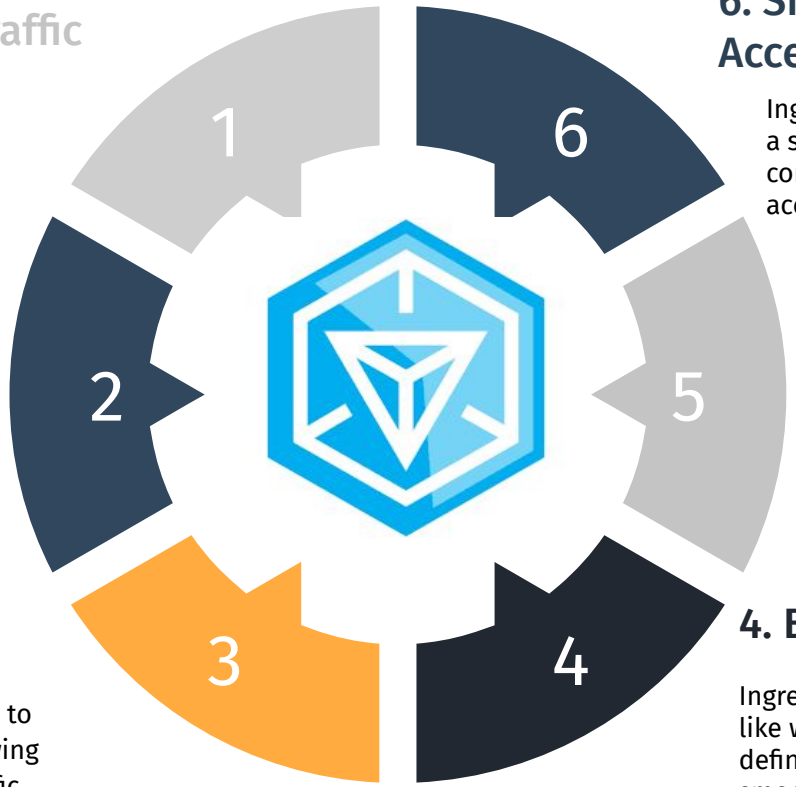
Ingress manages HTTP (port 80) and HTTPS (port 443) traffic, enabling secure and efficient routing to backend services across the Kubernetes cluster.

2. Path-Based Routing

Ingress allows precise routing of requests based on URL paths, such as /wear and /watch, directing traffic to the respective backend services.

3. Dynamic Configuration with ConfigMap

The Ingress Controller uses a ConfigMap to dynamically adjust Nginx settings, allowing custom configurations for handling traffic and optimizing performance.



4. Backend Service Integration

Ingress seamlessly integrates with services like wear-service and watch-service, defining port mappings and ensuring smooth communication between client requests and backend applications

5. Load Balancing

Ingress facilitates load balancing across multiple backend pods, ensuring high availability, scalability, and improved performance of services within the Kubernetes environment.

6. Simplified External Access

Ingress combines multiple services under a single external endpoint, reducing complexity and providing streamlined access to cluster resources for end users.

Components

ConfigMaps

Externalize configuration data dynamically for applications without modifying container images or redeploying.

Deployments

Ensure scalable, high-availability applications by managing desired Pod replicas and their life cycles.

Services

Provide consistent networking and load balancing between Pods and expose them to external clients.

Ingress

Direct HTTP/HTTPS traffic to backend services, simplifying domain-based routing and SSL termination..

Service Accounts

Securely manage Kubernetes component access and resource permissions with role-based credentials.

Pods

Fundamental Kubernetes unit, hosting one or multiple tightly coupled containerized application instances.

Deployment YAML

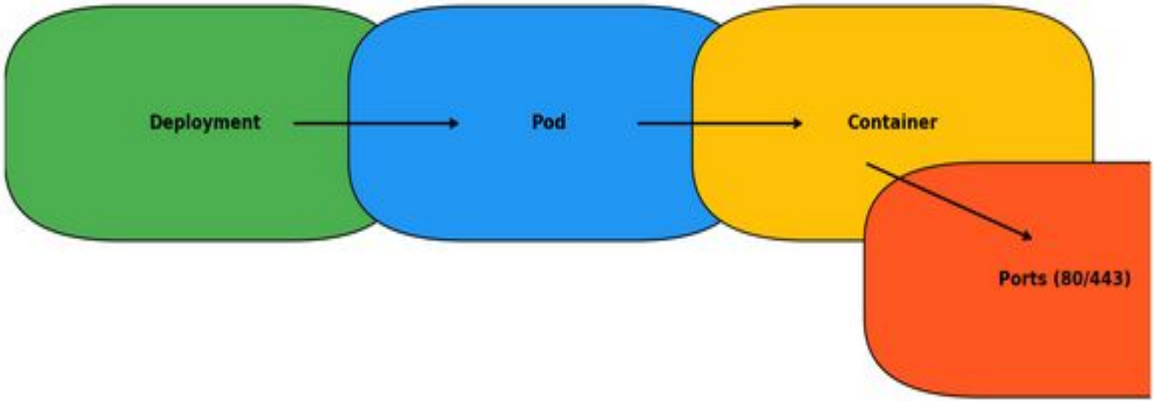
```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-ingress-controller
spec:
  replicas: 1
  selector:
    matchLabels:
      name: nginx-ingress
  template:
    metadata:
      labels:
        name: nginx-ingress
    spec:
      containers:
        - name: nginx-ingress-controller
          image: quay.io/kubernetes-ingress-controller/nginx-ingress-controller:0.21.0
          args:
            - /nginx-ingress-controller
            - --configmap=$(POD_NAMESPACE)/nginx-configuration
          env:
            - name: POD_NAME
              valueFrom:
                fieldRef:
                  fieldPath: metadata.name
            - name: POD_NAMESPACE
              valueFrom:
                fieldRef:
                  fieldPath: metadata.namespace
          ports:
            - name: http
              containerPort: 80
            - name: https
              containerPort: 443
```

Deployment Configuration: Defined as an apps/v1 Deployment with a replica count ensuring one pod, offering scalability options.

Container Details: Runs nginx-ingress-controller (image: quay.io/...:0.21.0) with dynamic config via --configmap=$(POD_NAMESPACE)/nginx-configuration.

Environment & Ports: Dynamically sets environment variables POD_NAME and POD_NAMESPACE, and exposes HTTP (80) and HTTPS (443) ports for traffic handling.

Deployment of Nginx Ingress Controller



Deployments

Represents the configuration managing replicas.

Pods

Holds the containerized application.

Container

Runs the Nginx Ingress Controller.

Ports

Exposes HTTP (80) and HTTPS (443) for traffic routing.

Service YAML

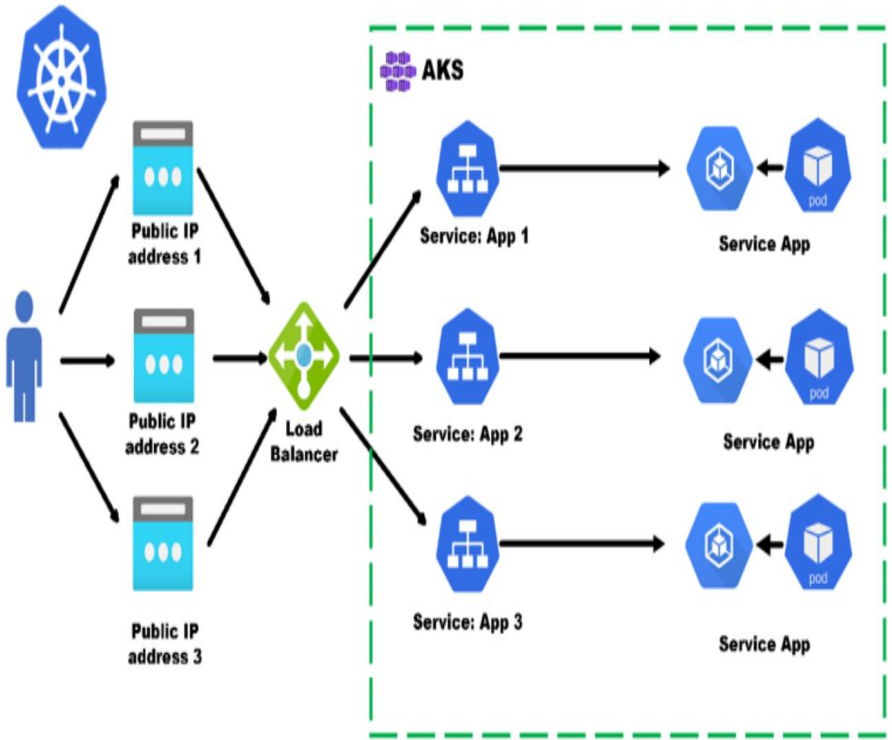
```yaml
apiVersion: v1
kind: Service
metadata:
  name: nginx-ingress
spec:
  type: NodePort
  selector:
    name: nginx-ingress
  ports:
  - port: 80
    targetPort: 80
    protocol: TCP
    name: http
  - port: 443
    targetPort: 443
    protocol: TCP
    name: https
```

Service Type:The NodePort type exposes the service externally via `<NodeIP>:<NodePort>`, enabling external Ingress Controller access.

Selector: Ensures the service targets `nginx-ingress` pods, accurately routing external traffic to the Nginx Ingress Controller within the cluster.

Ports: Defines port 80 for HTTP and 443 for HTTPS, enabling Ingress Controller to route traffic via NodePort service.

Ingress Service



1. User Requests Route Through Load Balancer

2. Load Balancer Distributes to Kubernetes Services

3. Kubernetes Services Manage Pods

Resource YAML

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ingress-wear-watch
spec:
  rules:
  - http:
      paths:
      - path: /wear
        pathType: Prefix
        backend:
          service:
            name: wear-service
            port:
              number: 80
      - path: /watch
        pathType: Prefix
        backend:
          service:
            name: watch-service
            port:
              number: 80
```

Rules: Specifies HTTP routes to direct /wear to wear-service and /watch to watch-service, ensuring targeted traffic flow.

Paths: Configures distinct paths /wear and /watch, enabling precise routing of external requests to designated backend services.

Backend Services: Maps wear-service and watch-service with port 80, facilitating seamless communication between Ingress and service backends.

Kubernetes Commands

kubectl get pods

Lists all Pods running in the current namespace.

kubectl describe pod

Displays detailed information about a specific Pod's configuration.

kubectl apply -f

Applies configuration changes defined in a YAML or JSON file.

kubectl delete pod

Deletes the specified Pod from the current namespace.

kubectl logs

Fetches logs from a specified Pod for debugging purposes.

kubectl exec -it

Executes a command inside a running Pod's container interactively.

Key Takeaways

Enhanced Traffic Management

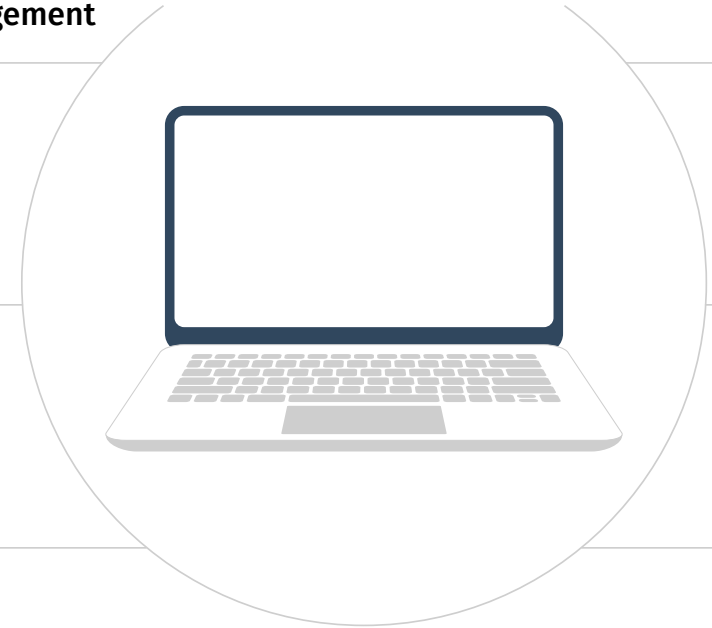
Successfully routed external traffic to internal services using Ingress configurations.

Simplified Deployment

Leveraged YAML files to automate Kubernetes component setup and resource management.

Improved Scalability

Demonstrated the ability to scale applications dynamically based on demand.



Secure Access Control

Used Service Accounts to manage secure and efficient permissions for Kubernetes resources.

Streamlined Service Exposure

Exposed multiple microservices (`wear-app` and `watch-app`) effectively through Services and Ingress.

Hands-On Kubernetes Experience

Gained practical insights into container orchestration and infrastructure management.

THANK YOU FOR YOUR VALUABLE TIME!