Course 7: Load Balancing & Autoscaling in Cloud-Managed Kubernetes Clusters

---

1. Introduction

In this course, we'll explore how **load balancers** manage traffic within a **cloud-managed Kubernetes cluster** and how they enable your **Task Manager** application to scale seamlessly.

You will learn:

- How internet traffic reaches your cluster and is distributed across pods.
- How autoscaling keeps your application performant.
- Different strategies for deploying load balancers, and trade-offs between them.

By the end of this course, you should understand how to expose your application securely to the internet, scale it efficiently, and justify your design choices.

---

2. Load Balancers in Managed Kubernetes Clusters

2.1 What Is a Load Balancer?

A **Load Balancer (LB)** distributes incoming requests across multiple backend instances (pods, nodes, instances, serverless services, etc) to:

- Ensure **high availability** (requests continue even if a pod fails).
- Enable traffic management with **horizontal scalability**.
- Improve **performance and fault tolerance**.

2.2 Load Balancer Integration Models

There are two main ways to integrate a load balancer with your managed Kubernetes cluster:

Option 1 — Kubernetes-Native Load Balancer

- Define a Kubernetes `Service` of type `LoadBalancer` or an **Ingress** resource.
- Kubernetes requests the creation of a **cloud-managed external load balancer** automatically via the **cloud controller manager** (depending on your Cloud provider, this automation may be not available).
- Backend targets are updated automatically whenever pods are added or removed.
- Health checks, firewall rules, and routing are automatically handled by the cloud provider.

Example:

```
apiVersion: v1
kind: Service
metadata:
  name: task-manager-service
spec:
  type: LoadBalancer
  selector:
    app: task-manager
  ports:
    - port: 80
      targetPort: 8080
```

✅ **Pros:**

- Fully integrated with Kubernetes.
- Automatic backend synchronization.

❌ **Cons:**

- Less control over advanced routing and policies.
- Provider-specific behaviors may be opaque.

Option 2 — Cloud-Managed Load Balancer

- You create the load balancer **directly in the cloud provider** (Terraform).
- Manually attach backend services that point to Kubernetes nodes or NodePorts.
- Define health checks, routing rules, SSL certificates, and security groups yourself.

✅ **Pros:**

- Full control over routing, SSL, and security policies.
- Easier to integrate Kubernetes with other non-Kubernetes services.

❌ **Cons:**

- More complex setup and maintenance.
- Automatic pod-to-backend synchronization may be not provided and requires additional scripting, depending on your cloud provider and configuration.

Guidance:
You can choose either approach. For your project, think carefully about the **pros and cons**, and be prepared to **justify your choice** during the project defense.

2.3 Cloud-Managed Load Balancer services

- **Google Cloud:** Cloud Load Balancing
- **AWS:** Elastic Load Balancer (ALB/NLB)

---

3. Load Balancer Placement in the Infrastructure

- The **external load balancer** sits in the **public subnet** of your VPC.
- Receives requests from the internet and forwards them to the **Kubernetes Service**.
- The Service distributes traffic to **application pods/nodes** running in your Node Pool.
- Your pods process requests independently and interact with the **managed SQL database** via its **private endpoint**.

---

4. Autoscaling

Managed clusters (GKE/EKS) allow horizontal autoscaling at two levels:

4.1 Horizontal Pod Autoscaler (HPA)

The HPA automatically adjusts the number of pods based on CPU, memory, or custom metrics.

Example:

```
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: task-manager-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: task-manager
  minReplicas: 1
  maxReplicas: 2
  metrics:
    - type: Resource
      resource:
        name: cpu
        target:
          type: Utilization
          averageUtilization: 70
```

- Pods are scaled up or down automatically.
- The load balancer automatically routes traffic to new pods.

⚠️ **Important:** For this project, do not rely solely on the HPA and scale only at the pod level. In a cloud-managed cluster, billing is based on **nodes** (active VM instances). Scaling only pods may improve **availability** but will **not optimize costs**.

Your cluster must also be able to **scale nodes automatically** using the **Cluster Autoscaler**.

💡 **Tip:** Choose **smaller node instances** (CPU & RAM) because you will most likely want to have only **one or two application pods per node**, and then scaling nodes. This ensures efficient resource usage and cost management.

4.2 Cluster Autoscaler

If all nodes are full and new pods cannot be scheduled, the **Cluster Autoscaler** automatically adds nodes to your Node Pool.
When resources are no longer needed, it scales down nodes to save costs.

> ⚠️ Node scaling may take a few minutes. During that time, your application must respond gracefully to overload, using **HTTP 429 – Too Many Requests**.

---

5. Integration with the Application

- Requests from the internet hit the **external load balancer**.
- The LB distributes requests to **multiple pods/nodes** in your cluster.
- Pods remain **stateless**; data is stored in the **managed SQL database**.
- Each request includes a **correlation_id** header for tracing and debugging.
- Requests are processed logically according to `request_timestamp`.

---

6. Best Practices

✅ Expose your app using **LoadBalancer Service** or **Ingress Controller**.
✅ Enable **HTTPS** with SSL certificates (cloud managed, self-signed, external provided).
✅ Use **readiness/liveness probes** to avoid routing traffic to unhealthy pods/nodes.
✅ Ensure full horizontal scaling using **HPA** and **Cluster Autoscaler**
✅ Return **HTTP 429** during overload.
✅ Monitor metrics from LB, pods, nodes and trace application requests.
✅ Keep your database private inside the VPC.

---

7. Key Takeaways

- **External Load Balancers** route internet traffic to your Kubernetes pods.
- You can choose **Kubernetes-native** or **cloud-managed** LB.
- Autoscaling at pod and node level ensures your application can handle traffic spikes.
- Your Task Manager application must be **secure, stateless, and scalable**.
- Be ready to **justify your load balancer and scaling strategies** during the project defense.
- You must be able to prove that your application can scale horizontally and follow the requirements during your defense => **Bring your own load testing script for the defense !**