Pod Resources

Resource Requests and Limits

Pod Resources Introduction

Kubernetes allows you to control how much CPU and memory a container can request or consume.

This is done using:

- **Requests**: Minimum resources the container is guaranteed

- **Limits**: Maximum resources the container can use

What Are Resource Requests and Limits?

Each container can specify:

- `requests`: The amount of CPU/memory **guaranteed**
- `limits`: The **maximum** CPU/memory allowed

If a container exceeds its memory limit, it is killed.
If it uses more CPU than its limit, it's throttled.

CPU and Memory Requests

- **CPU** is measured in millicores (1000m = 1 core)
- **Memory** is specified in Mi or Gi

```
resources:
  requests:
    cpu: "100m"
    memory: "64Mi"
```

This means the container is guaranteed 0.1 CPU and 64Mi of RAM.

CPU and Memory Limits

```
resources:
  limits:
    cpu: "500m"
    memory: "256Mi"
```

This sets the upper bounds:

- CPU cannot exceed 0.5 cores
- Memory cannot exceed 256Mi, or the container will be killed

Why Set Resource Constraints?

- Prevent a noisy neighbor from consuming all node resources
- Ensure predictable performance
- Improve scheduling decisions
- Avoid out-of-memory (OOM) crashes

Best Practices

- Start with realistic baseline requests
- Don’t set limits too close to requests
- Monitor usage and adjust over time
- Use `LimitRange` and `ResourceQuota` in namespaces

Example Scenario

A web server with:

- Minimal traffic most of the time
- Occasional bursty load

```
resources:
  requests:
    cpu: "100m"
    memory: "64Mi"
  limits:
    cpu: "500m"
    memory: "256Mi"
```

Provides guaranteed capacity while handling spikes up to a safe limit.

Conclusion

- Resource requests and limits are essential for stability and fairness
- Right-sizing is key to efficiency and cost control
- Use monitoring tools to fine-tune your resource profiles