KUBERNETES PATTERNS

PREDICTABLE DEMANDS

Kubernetes can manage applications written in different programming languages as long as the application can be run in a container. However, different languages have different resource requirements. Typically, a compiled language runs faster and often requires less memory compared to just-in-time runtimes or interpreted languages.
Considering that many modern programming languages in the same category have similar resource requirements, from a resource consumption point of view, more important aspects are the domain, the business logic of an application, and the actual implementation details.
Besides resource requirements, application runtimes also have dependencies on platform-managed capabilities like data storage or application configuration.

Runtime Dependencies

Dependency on a PersistentVolume

```
apiVersion: v1
kind: Pod
metadata:
name: random-generator
spec:
containers:
- image: k8spatterns/random-generator:1.0
name: random-generator
volumeMounts:
- mountPath: "/logs"
name: log-volume
volumes:
- name: log-volume
persistentVolumeClaim:
claimName: random-generator-log
```

The **scheduler** evaluates the kind of volume a Pod requires, which affects where the Pod gets placed. If the Pod needs a volume that is not provided by any node on the cluster, the Pod is not scheduled at all. Volumes are an example of a runtime dependency that affects what kind of infrastructure a Pod can run and whether the Pod can be scheduled at all.

Runtime Dependencies

Dependency on a Configmaps or Secrets

```
apiVersion: v1
kind: Pod
metadata:
  name: random-generator
spec:
  containers:
  - image: k8spatterns/random-generator:1.0
    name: random-generator
  env:
  - name: PATTERN
    valueFrom:
      configMapKeyRef:
        name: random-generator-config
        key: pattern
```

Configurations are another type of dependency. Almost every application needs some configuration information, and the recommended solution offered by Kubernetes is through ConfigMaps. Your services need to have a strategy for consuming settings either through environment variables or the filesystem. In either case, this introduces a runtime dependency of your container to the named ConfigMaps. If not all of the expected ConfigMaps are created, the containers are scheduled on a node, but they do not start up.

Resource Profiles

Dependency on a Resource Profiles

```
apiVersion: v1
kind: Pod
metadata:
  name: random-generator
spec:
  containers:
  - image: k8spatterns/random-generator:1.0
    name: random-generator
   resources:
     requests:
       cpu: 100m
       memory: 200Mi
     limits:
       memory: 200Mi
```

- Memory
- Cpu
- ephemeral-storage

Compute resources in the context of Kubernetes are defined as something that can be requested by, allocated to, and consumed from a container. The resources are categorized as compressible (i.e., can be throttled, such as CPU or network bandwidth) and incompressible (i.e., cannot be throttled, such as memory).
Making the distinction between compressible and incompressible resources is important. If your containers consume too many compressible resources such as CPU, they are throttled, but if they use too many incompressible resources (such as memory), they are killed (as there is no other way to ask an application to release allocated memory).

Resource Profiles

Depending on whether you specify the requests, the limits, or both, the platform offers three types of Quality of Service (QoS):

Best-Effort
Pods that do not have any requests and limits set for its containers have a QoS of Best-Effort. Such a Best-Effort Pod is considered the lowest priority and is most likely killed first when the node where the Pod is placed runs out of incompressible resources.

Burstable
A Pod that defines an unequal amount for requests and limits values (and limits is larger than requests, as expected) are tagged as Burstable. Such a Pod has minimal resource guarantees but is also willing to consume more resources up to its limit when available. When the node is under incompressible resource pressure, these Pods are likely to be killed if no Best-Effort Pods remain.

Guaranteed
A Pod that has an equal amount of request and limit resources belongs to the Guaranteed QoS category. These are the highest-priority Pods and are guaranteed not to be killed before Best-Effort and Burstable Pods. This QoS mode is the best option for your application’s memory resources, as it entails the least surprise and avoids out-of-memory triggered evictions.

Pod Priority

```
apiVersion: scheduling.k8s.io/v1
kind: PriorityClass
metadata:
  name: high-priority
  value: 1000
globalDefault: false
description: This is a very high-priority Pod class
---
apiVersion: v1
kind: Pod
metadata:
  name: random-generator
  labels:
    env: random-generator
spec:
  containers:
  - image: k8spatterns/random-generator:1.0
    name: random-generator
  priorityClassName: high-priority
```

Pod priority allows you to indicate the importance of a Pod relative to other Pods, which affects the order in which Pods are scheduled.

Project Resources

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: object-counts
namespace: default
spec:
  hard:
    pods: 4
    limits.memory: 5Gi
```

```
apiVersion: v1
kind: LimitRange
metadata:
name: limits
namespace: default
spec:
  limits:
 - min:
     memory: 250Mi
     cpu: 500m
   max:
      memory: 2Gi
      cpu: 2
   default:
     memory: 500Mi
     cpu: 500m
   defaultRequest:
     memory: 250Mi
     cpu: 250m
   maxLimitRequestRatio:
     memory: 2
     cpu: 4
  type: Container
```

Working in a **shared** multitenant platform also requires the presence of specific boundaries and control units to prevent some users from consuming all the platform's resources.
One such tool is **ResourceQuota**, which provides constraints for limiting the aggregated resource consumption in a **namespace**. With ResourceQuotas, the cluster administrators can limit the total sum of computing resources (CPU, memory) and storage consumed.

Another helpful tool in this area is **LimitRange**, which allows you to set resource usage limits for each type of resource. In addition to specifying the minimum and maximum permitted amounts for different resource types and the default values for these resources, it also allows you to control the ratio between the requests and limits, also known as the overcommit level.

Project Resources

When several users or teams share a cluster with a fixed number of nodes, there is a concern that one team could use more than its fair share of resources.

Resource quotas are a tool for administrators to address this concern.

A resource quota, defined by a ResourceQuota object, provides constraints that limit aggregate resource consumption per namespace. It can limit the quantity of objects that can be created in a namespace by type, as well as the total amount of compute resources that may be consumed by resources in that namespace.

Resource quotas work like this:

- Different teams work in different namespaces. This can be enforced with RBAC.
- The administrator creates one ResourceQuota for each namespace.