Project Resources

- K8s is a self-service platform that enables to run applications on the designated isolated environments.
- However, working a shared multi-tenanted platform also requires the presence of specific boundaries to control units to prevent some users/pods consuming all the resources of the platform
- One such tool is ResourceQuota, which provides constraints for limiting the aggregated resources consumption in a namespace
- With ResourceQuotas, the cluster administrators can limit total sum of computing resources (CPU, memory) and Storage consumed. This can also limit total number of objects (Config Maps, Secrets, Pods or Services) created in a namespace
- Official documentation on Resource Quotas(https://kubernetes.io/docs/concepts/policy/resource-quotas/)
- Manifests with Resource Quotas.(https://github.com/asquarezone/ExpertKubernetes/commit/2179b771896aecd14812e7ccc916c2586c0096bb)
- Another useful tool is LimitRange, which allows setting resource usage limits for each type of resource. (https://kubernetes.io/docs/tasks/administer-cluster/manage-resources/memory-default-namespace/)

Type	Resource	Min	Max	Default Limit	Default Request
Container	CPU	100m	500m	500m	250m
Container	Memory	250Mi	2Gi	500 Mi	250 Mi

Health Probe

- This pattern is about how an application can communicate its health state to k8s
- Problem:
  - k8s regularly checks the container process status and restarts if issues are detected.
  - However, checking the process status is not sufficient to decide the health of the application.
  - For example,
    - a Java application might throw and OutOfMemoryError and still have the JVM Process running.
    - Application may freeze because it runs into infinite loop, deadlock
  - To detect these kind of situations k8s needs a reliable way to check the health of application
- Solution:
  - Let's see the checks k8s uses to detect and correct failures
  - Process Health Checks: This is simplest health kubelet constantly performs on the container process. If the container processes are not running, then container is restarted
  - Liveness Probe:
    - If your application runs into some deadlock, it is still considered healthy from the process health check's point of view.
    - To detect these kinds of issues or any other types of failures according to the application business logic k8s has a liveness probe.

- It is important to have a health check performed from outside rather than in application itself, as some failures may prevent the checks inside the application from reporting failure.

- Readiness Probe:
  - k8s has readiness probe.
  - The methods for checking are same as liveness (HTTP, TCP, EXEC), but corrective action is different, Rather than restarting the container a failed readiness probe causes the Container to be removed from service endpoint and will not receive any new traffic.

Liveness probe will restart container, but in readiness prob it removes container from endpoint.

Initialdelaysecond: it's time to wait for performing HC once container is up.

Exec: it used to perform action.