**Why *init-Containers* ?**

It’s sometimes necessary to prepare the main container which will be running our application or the main logic and its accomplished through init containers.

There can also be situations where we need to execute particular utilities or setup scripts that are not present in our main image ( as it would make it heavy and we only need it once in the beginning ).

Some of its properties are :

* It contains utilities or setup scripts not present in an app image ( making images light-weight)
* They always run to completion
* Init container executes sequentially and each of the init containers must succeed before the next can run.
* They support all the fields and features of app containers, including resource limits, volumes, and security settings.

### How init-container come to Action in Kubernetes!

* First, the **kubelet** will wait until the networking and storage are ready so that it can start running init containers.
* It then runs the Pod’s init containers in the order they appear in the Pod’s spec.
* Each init container must exit successfully before the next container starts.
* A Pod cannot be in **Ready** state until all init containers have succeeded.
* If the Pod **Restarts**, or is restarted, all init containers are executed again.

**Note** : Altering an init container image field is equivalent to restarting the Pod.

## Few Use Cases of init-container

* Init containers can contain utilities or custom code for setup that are not present in an app image.
* They can be given access to Secrets that app containers cannot access.
* Clone a Git repository into a Volume
* It can be used to wait for a service to start that is to be used by main app
* An init container is a good candidate for delaying the application initialization until one or more dependencies are available.

**How initContainers work**

* During Pod startup, the kubelet delays running init containers until the networking and storage are ready. Then the kubelet runs the Pod's init containers in the order they appear in the Pod's spec.
* Each init container must exit successfully before the next container starts. If a container fails to start due to the runtime or exits with failure, it is retried according to the Pod restartPolicy. However, if the Pod restartPolicy is set to Always, the init containers use restartPolicy OnFailure.
* A Pod cannot be Ready until all init containers have succeeded. The ports on an init container are not aggregated under a Service. A Pod that is initializing is in the Pending state but should have a condition Initialized set to true.
* If the Pod restarts, or is restarted, all init containers must execute again.

**How initContainers are different from normal Containers**

* Init containers support all the fields and features of app containers, including [resource limits](https://www.golinuxcloud.com/kubernetes-resources/), volumes, and security settings. However, the resource requests and limits for an init container are handled differently
* Also, init containers do not support lifecycle, livenessProbe, readinessProbe, or startupProbe because they must run to completion before the Pod can be ready.
* If you specify multiple init containers for a Pod, Kubelet runs each init container sequentially. Each init container must succeed before the next can run. When all of the init containers have run to completion, Kubelet initializes the application containers for the Pod and runs them as usual

### Restart Policy

The [restartPolicy](https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle/" \l "restart-policy) field of the pod defines on what condition should the containers within the pod restart themselves. There are three possible values: Always, OnFailure, and Never. When we set the restartPolicy of a pod to Always, the init containers’ restartPolicy will be set to OnFailure. This is because the Always policy restarts the containers even with a zero exit code.

However, the Always policy contradicts the expectation of an init container, which is to exit with a zero exit code. Therefore, **both the Always and OnFailure restartPolicy values set on the pod level will map to the OnFailure value for init containers**.

## 4. Lifecycle of Init Containers

Creating and running init containers are the first step for bringing a pod to life. Therefore, understanding the lifecycle of the init containers can help in understanding and debugging issues related to the readiness of a pod.

We’ll first look at the steps that an init container goes through, and then we perform a demonstration using a simple example.

### 4.1. Starting up the Init Containers

Firstly, the process begins when we create a pod resource on the Kubernetes cluster. The creation of the pod can be either directly using the Pod specification or indirectly through the Deployment, StatefulSet, or DaemonSet specification. Either way, the [kube-scheduler](https://kubernetes.io/docs/concepts/overview/components/" \l "kube-scheduler) will schedule the pod on a node. Then, the [kubelet](https://kubernetes.io/docs/concepts/overview/components/" \l "kubelet) process on the node will create and starts the init containers. **Due to the init containers’ sequential execution order, the process will run the first init container and observe its exit code before deciding on the next step**.

### 4.2. On Init Containers Failure

**When the init container returns a non-zero exit code, the startup process either fails the startup or restarts the process, depending on the restartPolicy of the pod**. If we set the restartPolicy to Never, the pod status will change to Failed, and the start-up process stops at this point.

Alternatively, for the restartPolicy value of Always or OnFailure, the whole process will be restarted, starting from the first init containers in the list. **Because of the possibility of multiple executions of init containers due to restarts, it’s important to design our init containers such that it’s**[**idempotent**](https://www.baeldung.com/cs/idempotent-operations).

Furthermore, if there’s a failure in the init containers, the status of the pod will turn to Init:Error. This tells us that the pod fails to startup due to errors in the init containers. When the same error occurs multiple times, the status will then turn into Init:CrashedLoopBackOff.