**Init Container**

## **🔰 What are Init Containers?**

**Init containers** are specialized containers that run **before** app containers in a Pod are started.

They are used to **perform initialization tasks** — such as setting up environment, copying files, checking prerequisites — that must be completed before the main application containers can start.

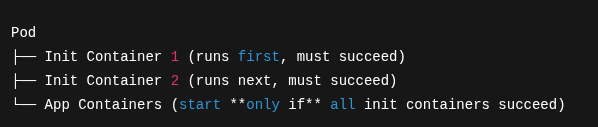
They always **run to completion**, and **each init container must complete successfully before the next one starts**. Only after all init containers complete successfully, the regular application containers are started.

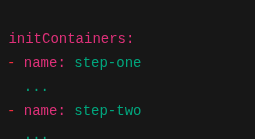
## **🧠 Why use Init Containers?**

| Use Case | Description |
| --- | --- |
| **Dependency checks** | Wait for a service or DB to be available before starting the main container |
| **Setting permissions** | Change file or directory permissions before the app runs |
| **Pre-pulling data** | Pull necessary configuration, keys, or binaries from an external source |
| **Environment setup** | Run scripts or tools to configure the environment |
| **One-time initialization logic** | Avoid baking logic into the main image by separating init tasks |

## **⚙️ Key Characteristics**

* Run **sequentially** and **only once**.





step-two will not start until step-one has exited successfully.

* Must **succeed** (exit code 0) to allow pod startup to proceed.
* **Main containers won’t start** until all init containers have successfully completed.
* If an init container **fails**, it will be **retried** based on pod restart policy.
  + If it repeatedly fails, it may trigger CrashLoopBackOff.
  + pod will be in Init:CrashLoopBackOff
* Run in the **same network and storage context** as the main containers.
* Can **share volumes** with app containers to pass data.
  + Pod status will be **Init:<x>/<y>** indicating progress.

## **🔹 Example YAML**

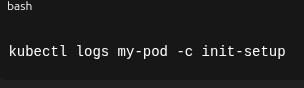
## 

## **🔹Init Containers vs Regular Containers**

| Feature | Init Containers | Regular (App) Containers |
| --- | --- | --- |
| Execution | Sequentially | Concurrently |
| Restart Policy | Restarted independently | Governed by Pod’s policy |
| Lifecycle Dependency | Must finish for app to start | Not dependent |
| Purpose | Setup/init logic | Main app logic |

## **🔹 Logging in Init Containers**

You can view init container logs just like regular containers:



For multiple init containers:



**📂 Shared Volumes between Init and App Containers**

* They write to shared volumes and allows us to **pass data between init containers and the main container**.



### Use Case:

* init-copy puts config files into an emptyDir volume.
* App container uses those files from /app/config.

## 🔹 How to Control Init Container Behavior

### With readinessProbe?

* **Not supported** in init containers — they are **run-to-completion**, not long-running.

### With resources, env, securityContext?

* You can define all those just like in regular containers.

## **🔹 Summary Table**

| Topic | Description |
| --- | --- |
| Purpose | Prepare environment for main containers |
| Runs before | App containers |
| Sequence | Ordered, one-by-one |
| Restarts on failure | Yes |
| Logging | kubectl logs POD -c init-container |
| Volume sharing | Supported via shared volumes |
| Different images/tools | Yes |
| Readiness/Liveness probes | Not applicable |
| Resource management | Supported |
| SecurityContext | Supported |