Here is a list of five good practices to make your CI/CD pipeline more efficient:

### Execution of resources

You need to ask yourself: What kind of resources does your workflow consume the most? Does your application need to be tested on specific operating systems like Linux, macOS, or Windows?

If you can answer those questions, you can then choose appropriate resources. CI/CD tools provide us with virtual machines with default settings, but maybe you need more CPU or RAM or GPU. In that case, you should use a custom VM for your computation-heavy workloads.

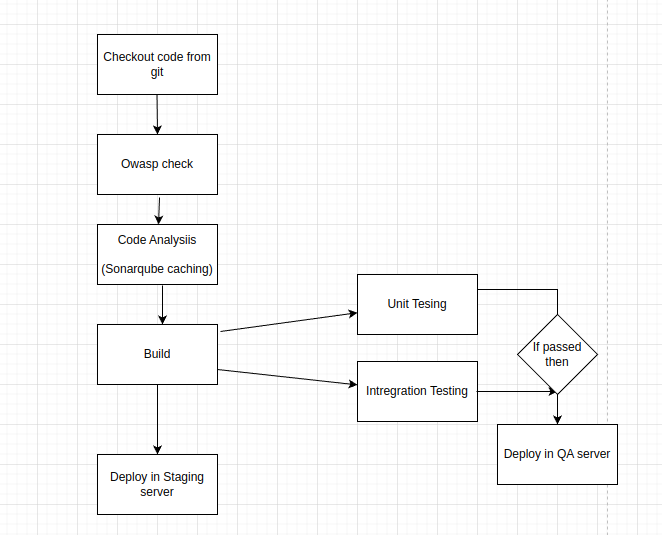
### Docker image and layer caching

If you containerize your application with Docker, then choosing the right image for your project can have a huge impact on your build time. Depending on the language you use to build your application, there are already basic images created for it with dependencies and tools included.

But maybe you do not need all of them, or you have other requirements. Then you should consider creating your own image according to best practices directly from tool creators.

### Parallelism

If you run your workflows, jobs, and tests in a linear fashion, run them simultaneously instead. You can use a matrix strategy to dispatch jobs multiple times with different settings. You can split tests over a range of dedicated environments and run them at the same time.



Depending on whether your tests are isolated or not, you will have to use a different approach. Some tests might depend on each other, some tests might need access to a database or other services. You need to take that into account. A basic rule of thumb is what can run in parallel, should run in parallel.

### Caching

As a first step, you should examine your workflows and the jobs that are run inside them. If any of these jobs require additional dependencies or files, this is where you likely can use caching.

There is no need to download the same package version all over again during a job.

You can cache it and use it in your job instead of downloading it each time. You can also use it for your build or pre-build artifacts. While using caching, you need to remember that it is possible to create race conditions in caching across jobs in a workflow, and it can also break isolation between jobs.

### Optimize tests

As your application grows, the number of manual and automated tests will increase and should increase, and you should follow these guidelines to have them under control:

* Split your tests into unit tests, integration tests, e2e tests, smoke, acceptance, and security tests.
* Run them only when it is necessary and not on each run of the pipeline.
* Identify slow, duplicated, or flaky tests, and improve or delete them if necessary.
* Use a profiling tool to check every change before and after.
* Mock external services in your unit tests.
* Run the database in memory for unit tests if possible.