Name/reference	Description	Example	Fix	Suggestion	Good/Bad Practice
Outdated dependencies S28 S69 S46 S49 S70 S69 S72 S32 S31 S6 S85-90 S99-110	The overuse of low-level general abstraction s such as file and package tools is considered an anti-pattern.	Outdated GPG key[1]	Removing the old keys → Updating to newer version	manual or external updates outside automated IaC workflows or manually crafted base images → use Docker files	Using non reproducible images and environments Distribution assumption for every machine.
Version specific installation/declaration S14 S29 S22 S4 S6 S7 S13 S61 S19 S23 S26 S27 S28 S32 S33 S8 S69 S50 S51 S46 S70 S28 S85-90 S99-110	When a single version of a library/pack age is declared → outdated → lead to errors	Changed dependen cies Version mismatch [2]	Update and install the latest version of the library/packag e	Declare a range of versions that are compatible with the rest of the dependencies.	Understand compatible dependency versions Installing/declaring “Latest” → anti-pattern Use the correct format of dependency pinning. Document the initial version of the packages and their source. Create Dockerfiles for the specific requirements
Hardware specific commands	Executing commands	couldn’t select	Install the compatible		Document the initial version of the hardware and

S69 S70 S71	for a specific kind of hardware → missing the hardware can lead to errors.	device driver \ with capabilities: [gpu] [3]	version with the available hardware.		configurations that are integrated together.
General Assumptions about the environment S72 S70 S71 S14 S29 S22 S4 S6 S7 S13 S61 S19 S23 S26 S27 S28 S32 S33 S8 S69 S50 S51 S46 S70 S28 S85-90 S99-110				Assume that the code will be executed by others in a fresh session which means that they will not have access to artifacts or customizations from the original environment	Using absolute path instead of relative path inside the project directory setting a working directory in your script → use paths relative to the home of the project Distribution assumption for every machine
Missing dependencies S22 S7 S20 S28 S9 S69 S72	When dependencies don't exist, even without errors, they may cause improper output.	Produce empty perf files[4]	Installing the package correctly.	Include all the installation commands. use Docker files	Default package/library assumption → Bad Document all the packages/libraries needed for working. Ensure the reproducibility package is complete and file

S70					paths are not specific to your machine
Violation of Idempotence S57 S54 S65 S49 S48 S55 S51 S57 S70 S72 S31 S6 S85-90 S112-119	When multiple executions won't end in the same state in configuration.	Duplicate lines in source.list file[5] Creating files without checking the existence or the path.	Removing the duplicated lines from the files.	Check for already installed packages and do not install them on every run	command and shell that execute ad-hoc operating system commands can break idempotence. → execution should be guarded using conditionals that check the state of the infrastructure and its components Docker and Docker compose files simplifies cloning, sharing, and versioning environments Skipping tasks in Ansible plays is not recommended.
Package specific commands S69 S70 S72 S57	Executing commands for a specific package→ missing the package can lead to errors.	Syntax errors executing a bash script → not running with the specified bash [6]	Run using the correct package	Complete running command	If you use user-written commands, include the exact version in your reproducibility package