Global Functional Requirement Overrides

Jump to navigation Jump to search

Every component for every distributed system created by SILO GROUP has a set of functional requirements that must be met which override any functional requirements specified in that system's design. These shared global functional requirements are called global functional requirement overrides.

Contents

1 Global FR Overrides
- 1.1 Codebase
- 1.2 Dependencies
- 1.3 Configuration
- 1.4 Dependent Services
- 1.5 SDLC Practices
  - 1.5.1 Compilation, Deployment, Operationalization
  - 1.5.2 Release Management
- 1.6 Process Behaviour, Integration, Concurrency
- 1.7 Disposability
- 1.8 Environments
- 1.9 Logging
- 1.10 Paths, Working Directories

Global FR Overrides

GFRO's are largely sourced and modified from "The 12 Factor App". See the 12 factor app website for more details.

While many of the concepts from 12 factors are used, they are still listed explicitly here as opposed to linking to the external site. You will see another reason for this is due to quite a few strong deviations from them that do meet the spirit of "12 Factor Application" but are improvements on their publications' specifications to meeting their goals with modern linux systems in mind instead of tailoring to a single specific albeit unwritten container product.

As of the time of this writing, many of these are actually GNFRO's that need moved over to a dedicated page just for those. This is an ongoing study.

Codebase

GFRO_2020-03-01_1	1:1 ratio between the application and the codebase.

Dependencies

GFRO_2020-03-01_2	Full dependency tree is provided for the component/system and checked for at build time / test time. This implies explicit dependency declaration in a dependency manifest, even where not demanded by the build system.
GFRO_2020-03-03_3	No common dependencies are assumed for any library, tool, or external executable.

Configuration

GFRO_2020-03-01_4	A configuration file must exist for the application.
GFRO_2020-03-01_5	Anything that is to vary between environments is to be included or accounted for in the configuration file such that no code changes are required when moving to new environments.
GFRO_2020-03-01_6	A varying item can only be removed from the configuration file by checks in the code of the application itself. This does not imply that environment-specific code is allowed. This is purely for the purpose of environment aspect detection by the application in abstraction operations.
GFRO_2020-03-01_7	Configuration files will be in an object notation format, such as INI, JSON, XML, YAML, or otherwise.
GFRO_2020-03-01_8	Runtime-processed or compiled code in the source are not to be used for configuration.
GFRO_2020-03-01_9	Strict separation between code and copnfiguration will be maintained at all times.
GFRO_2020-03-01_10	Environment variables provided by the environment can be used dubiously when the situation is appropriate, but no new environment variables are to be introduced to the system for the component/system.
GFRO_2020-03-01_11	Configuration files are not to be considered part of the code. Example configuration files may be included with the source for the purpose of consumers creating new configuration files, but deployments are not to be done with configuration files bundled in the source.

Dependent Services

GFRO_2020-03-01_12	Services that the application or system communicates with are to be decoupled in such a way as to be agnostic of how those services are interacted with; no deployment-specific behaviour is to be introduced.
GFRO_2020-03-01_13	DNS addresses are to be used to identify dependent services not on the same host and accessed over the network.
GFRO_2020-03-01_14	Interaction should be developed in such a way where the same type of service can be exchanged without breaking the application's ability to exchange with the resource. This does not exclude setup steps being necessary such as database population.
GFRO_2020-03-01_15	The application will be agnostic to clustering forms of its attached resources (for example, should not be aware of 2 mysql servers in a sharding mode, it would view the resource as a MySQL server and interact with the master).

SDLC Practices

These are not functional requirements but are defined here to guide the SDLC.

Compilation, Deployment, Operationalization

1. Build, deploy, run are to be 3 distinct phases. These phases may be triggered seamlessly by an orchestration/automation system, but must be built separately and managed by a controller of some sort.
2. Unit tests will take place as part of the build phase.
3. Functional tests will take place as the last part of a deploy phase.
4. The deploy phase will place configuration files in the target environment that properly correspond to the version or release candidate.

Release Management

1. Once the codebase reaches an arbitrary milestone, the codebase revision can be tagged as a release candidate.
2. Every release candidate will be given a unique identifier representing the date and number of the RC (YYYY-MM-DD_RC_##). The RC identifier will be paired to its corresponding codebase revision using GIT TAGS in the repository.
3. Once a release candidate meets release criteria specified by the architect, a deliberation over whether it is approved as a release version or not takes place. The final approver/rejector is the architect.

4. Once approved, a release candidate will be given an incrementing version number as V_##.##, representing the release's primary version and minor version number. Major structure updates will be given an increment to the major version number, changes to existing design will be given minor version numbers. The version number will be paired to its corresponding release candidate using GIT TAGS.
5. No codebase revision not tagged as a release candidate will be considered to be made a release.
6. Minor versions revert to 0 when major revisions are incremented.

Process Behaviour, Integration, Concurrency

GFRO_2020-03-01_16	Distributed Web System components are stateless and share-nothing with exception to intrinsically stateful components, such as databases and object storage.
GFRO_2020-03-01_17	Persistent services will be integrated with systemd via a unit file for service state control.
GFRO_2020-03-01_18	All components will have a command line interface meeting IEEE STD 1003.1-2017, chapter 12.
GFRO_2020-03-01_19	Processes will under no circumstances place cache generation in the build phase for the component and will create/consume caches during runtime.
GFRO_2020-03-01_20	Stateful information will never be cached or held in memory.
GFRO_2020-03-01_21	On persistent compute resources, persistent services that receive purely HTTP responses will be configured to listen on loopback addresses and be fronted by dedicated HTTP services as a well-known HTTP server such as Apache or NGINX, relaying traffic through a reverse proxy configuration. If the environment is a container, these can listen to externally-listening addresses, so long as traffic to the persistent service is being relayed by the Apache or NGINX pool.
GFRO_2020-03-01_22	SSL will be handled purely by the well-known HTTP server.
GFRO_2020-03-01_23	Self-signed certificates are never used for externally-accessible services.
GFRO_2020-03-01_24	Private keys and their corresponding certificates will be shared between all load-balancing endpoints acting as a reverse proxy for a given persistent service.
GFRO_2020-03-01_25	The component will be *agnostic* of the fronting well-known HTTP relay service.
GFRO_2020-03-01_26	Processes will be *agnostic* of scaling determinations or the spawning of additional instances for load scaling achieved through either orchestration abstraction or dependent service decoupling.
GFRO_2020-03-01_27	Scaling of applications will be done with a layer that the application is not dependent upon.
GFRO_2020-03-01_28	Processes will never daemonize and rely solely on systemd for service state control. They will use the systemd unit to manage lock files, PID files et al.
GFRO_2020-03-01_29	Processes will not use port forwarding or relays on the host through the systemd functions. Only service state control features (start/stop/restart) of systemd are used.

Disposability

GFRO_2020-03-01_30	Processes should strive to minimize startup time.
GFRO_2020-03-01_31	Startup method should be via systemctl
GFRO_2020-03-01_32	Processes should stop gracefully when revieving a SIGTERM.
GFRO_2020-03-01_33	Preferred stop method should be via systemctl.
GFRO_2020-03-01_34	Worker processes should return pending tasks to the work queue.
GFRO_2020-03-01_35	All processes should strive for transactions to be reentrant or buffered.

Environments

GFRO_2020-03-01_36	Environments **will** be identical and kept in parity. Disparity in environments should be treated as a defect in the environment and prompt an effort to regenerate the environment.
GFRO_2020-03-01_37	DNS and compute resources will be used to segregate and identify environments.
GFRO_2020-03-01_38	Environments will be regularly regenerated using Infrastructure-as-Code practices.
GFRO_2020-03-01_39	Environment regeneration will be decoupled from application deployment.
GFRO_2020-03-01_40	Configuration deployment will be decoupled from application deployment even where triggered by the same mechanism.
GFRO_2020-03-01_41	Continuous delivery patterns will be used for deployment in an automated fashion.
GFRO_2020-03-01_42	Under no circumstances will deployment to production be triggered automatically outside of environment regeneration.
GFRO_2020-03-01_43	Deployment is triggered after environment regeneration.
GFRO_2020-03-01_44	Deployment to lower environments is automatic after each successful merge of a release candidate tag to the master branch.
GFRO_2020-03-01_45	Deployment automations will be decoupled from service state control, and will trigger appropriate actions during deployments.

Logging

GFRO_2020-03-01_46	SILO Applications will write all logs to STDOUT/STDERR.
GFRO_2020-03-01_47	The systemd unit for persistent services will redirect STDOUT and STDERR to syslog using the StandardOutput=syslog, StandardError=syslog directives.
GFRO_2020-03-01_48	The application will not be otherwise logging-aware unless the component's purpose specifically calls for it.
GFRO_2020-03-01_49	The SyslogIdentifier in the systemd unit file will be set to the high level name of the application excluding any special characters or whitespace.
GFRO_2020-03-01_50	User and group owner of the process will be set in the systemd unit file using their corresponding directives.

Paths, Working Directories

GFRO_2020-03-01_51	All paths to external files should be configurable with exception to the configuration file where these paths are specified.
GFRO_2020-03-01_52	Default paths can be assigned, such as the configuration file.
GFRO_2020-03-01_53	The working directory should be set in the systemd unit file.
GFRO_2020-03-01_54	The configuration file should be searched for at /etc/[${system}/]${component}/config.ini for deployed systems
GFRO_2020-03-01_55	The fallback configuration file path should be at ${HOME}/.config/[${system}/]${component}/config.ini and this should be preferred for local testing.
GFRO_2020-03-01_56	The application should find its configuration file regardless of which directory the application is placed in.
GFRO_2020-03-01_57	The application should all other paths specified in the configuration file regardless of which directory the application is placed in.
GFRO_2020-03-01_58	The application should function normally regardless of which directory the application is placed in.
GFRO_2020-03-01_59	The current working directory at the execution of the application should be irrelevant to the program's functioning.

Retrieved from "https://wiki.silogroup.org/index.php?title=Global_Functional_Requirement_Overrides&oldid=240"

This page was last edited on 1 August 2020, at 22:08.

Content is available under Attribution-NonCommercial-NoDerivatives 4.0

International unless otherwise noted.