DIS Documentation

Release 1.0.0

ATSI S.A.

Nov 08, 2017

Contents

Download the checklist poster here!

A checklist for containers - 11 things that will make your app production-ready.

CHAPTER 1

1. Explicitly declared and isolated dependencies

- Use container descriptors (e.g. a `Dockerfile`).
- Store descriptors with the codebase.
- Select a proper base image.
- Minimize usage of system-wide packages - make use of `rubygems/pip/nuget/npm/yarn/apt-get/yum/apk` instead.
- Use ONLY those packages that are actually needed.

CHAPTER 2

2. One stateless and share-nothing process

- Make no disctinction between local and third-party services.
- Treat all backing services as attached resources.
- Don't run your app in a container along with datastores (such as MySQL or CouchDB), messaging/queueing systems (such as RabbitMQ or Beanstalkd), email services (such as Postfix) or caching systems (such as Memcached).
- Don't assume the local file system is permanent.
- Don't keep session state in your application.

CHAPTER 3

3. Configuration stored in the environment

- Make your app to use environment variables out of the box.
- Define default values for variables, fail when it's required and there is no value.
- Provide a documentation file describing how to configure the container – simple `readme.md` will do.
- Allow to mount configuration files as a volume – if modification of the app is not possible.

CHAPTER 4

4. Services exported via port bindings

- Don't rely on runtime injection of a webserver into the execution environment to create a web-facing service - make apps, not libraries.
- Embed a webserver in your app such as Tornado for Python, Thin for Ruby, Jetty/Tomcat/Undertow for Java.
- Reuse webservers such as nginx or apache as part of your app.
- Explicitly expose ports in the container image manifest file.
- Remember that an external proxy is a thing – if your app exposes HTTP(s) binding, it should be prepared to have it's base address rewritten (e.g. passed as env vars).

CHAPTER 5

5. Fast startup and graceful shutdown

- Startup should take a few seconds – counting from the time the launch command is executed until the process is up and ready to receive requests or jobs.
- React on a `SIGTERM` signal – perform a graceful shutdown and free all resources (release locks, close ports etc.).
- Be robust against sudden death – handle unexpected, non-graceful terminations (such as clients disconnect or timeouts).

CHAPTER 6

6. Logs treated as event streams

- Treat logs as streams and write them unbuffered, to `stdout` or `stderr`.
- Never concern your app with routing or storage of its output stream.
- Don’t write logs to the file system.
- Don’t attempt to manage logfiles.

CHAPTER 7

7. Reasonably small size

- Use a smaller base image. Alpine is the way to start. Most likely, there are alpine tags for the programming language you are using.
- Minimize layers by combining your *RUN* statements.
- Fine-tune your package manager to install only what's needed with no overhead. Use commands like `apt-get --no-install-recommends`, `apk add --no-cache`.
- Clean up after installing packages in the same layer. Use commands like `rm -rf /var/lib/apt/lists/*`, `yum clean all`.
- Don't install debug tools (like `vim/curl`) in your production images.
- Verify the quality of your images – lint your `Dockerfile` in external tools such as http://www.fromlatest.io.

CHAPTER 8

8. Verifiable readiness and health status

- Expose health checking endpoints.
- Use HTTP endpoints where possible. (such as those exposed by Spring Boot Actuator).
- Provide an additional shell scripts as part of your image when HTTP is not an option.

CHAPTER 9

9. Discoverability

- Give meaningful content where it is expected – when the container exposes port `80` or `8080`, it is expected that hitting the base URL will deliver features, or at least a guide/hyperlink telling how to get to the features.
- Use standard port bindings for standard protocols.
- Deliver mature REST APIs – define proper resources, HTTP verbs and hypermedia controls. Refer to Richardson's REST maturity model.
- Use Swagger for API description and expose the definition in your app.
- Embed a Swagger UI right in your application (e.g. by using SpringFox library).

CHAPTER 10

10. Unidirectional, declarative deployment

- Be as declarative as possible when describing your deployment – prefer deployment descriptors over shell scripts (such as `docker-compose` .yaml files, kubernetes .yaml descriptors etc.).
- Keep your deployment versioned – e.g. in a git repository.
- Don't mutate releases/deployed apps – make sure there are no additional manual steps and no tampering with containers once they are deployed.
- Use rollbacks if you want to return to previous versions.
- Enforce a strict, one-way separation between the build, release, and run stages as it is impossible to make changes to the code at runtime and to propagate those changes back to the build stage.
- Ship admin code with application code to avoid synchronization issues (things such as database migrations, other one-time scripts).
- Run admin/management tasks as one-off processes.

CHAPTER 11

11. Dev/Prod parity

- Make the time gap small - write some code and have it deployed hours or even just minutes later (or fight for it if it's still impossible in your organization!)
- Make the personnel gap small – if you wrote the code, get involved in deploying it and watching its behavior in production!
- Make the tools gap small - keep development and production as similar as possible in form of preferably the same declarative descriptors.
- Don't use different backing services between dev and prod – if it's Postgres on prod, why it's SQLite on dev?
- Use declarative provisioning tools such as Puppet, Ansible, Chef along with lightweight virtual environments like Vagrant to run local environments which closely approximate production
- Give a great zero-to-dev experience – preferably a one-liner (like `vagrant up` or `docker-compose up`) to spin the whole thing up. A `readme.md` comment wouldn't kill you either.