CONSULTING.TECHNOLOGY.OUTSOURCING

Designing Applications for Cloud Foundry

People matter, results count.

Objectives of CF – Design Applications

- Purpose:
  - To learn pivotal cloud foundry 12-Factor Applications
- Product:
  - 12-Factor Applications
  - Design Guidelines
- Process:
  - To learn 12-Factor Applications and design guidelines of Cloud Foundry to develop Cloud Applications.

Table of Contents

- 12-Factor Applications
- Design Guidelines
  - Application architecture concerns:
    - Load Balancing / Session Management
    - Local file system
    - Port Limitations

12-FACTOR APPLICATIONS

12-Factor Application

- http://12factor.net

- Outlines architectural principles for modern apps
  - Focus on scaling, continuous delivery, portable and cloud ready

12-Factor Application

I. Codebase
One codebase tracked in SCM, many deploy

II. Dependencies
Explicitly declare and isolate dependencies

III. Configuration
Store config in the environment

IV. Backing Services
Treat backing services as attached resources

V. Build, Release, Run
Strictly separate build and run stages

VI. Processes
Execute app as stateless processes

VII. Port binding
Export services via port binding

VII. Concurrency
Scale out via the process model

IX. Disposability
Maximize robustness with fast startup and graceful shutdown

X. Dev/ prod parity
Keep dev, staging, prod as similar as possible

XI. Logs
Treat logs as event streams

XII. Admin processes
Run admin/ mgmt tasks as one-off processes

12- Factor Application

I. Codebase
One codebase tracked in SCM, many deploy

II. Dependencies
Explicitly declare and isolate dependencies

III. Configuration
Store config in the environment

- Codebase
  - An application has a single codebase
    - Multiple codebase = distributed system (not an app)
      - Each component in a codebase can (should) be an app
    - Tracked in version control
      - Git, Subversion, Mercurial, etc.
    - Multiple deployments
      - i.e development, testing, staging, production, etc.

12- Factor Application

I. Codebase
One codebase tracked in SCM, many deploy

II. Dependencies
Explicitly declare and isolate dependencies

III. Configuration
Store config in the environment

- Dependencies
  - Packaged as jars (java), RubyGems, CPAN (Perl)
  - Declared in Manifest
    - Maven POM, Gemfile / bundle exec, etc.
  - No reliance on specific system tools
    - i.e Linux tool not available on windows

12- Factor Application

I. Codebase
One codebase tracked in SCM, many deploy

II. Dependencies
Explicitly declare and isolate dependencies

III. Configuration
Store config in the environment

- Configuration
  - Separate from the code
  - Also separate from the application
    - i.e. DB credentials; hostnames, passwords
    - Acid test – could the code base be made open source?
    - Internal writing (i.e. Spring Configuration) considered part of codebase.
  - Environment variables recommended.

12- Factor Application

IV. Backing Services
Treat backing services as attached resources

V. Build, Release, Run
Strictly separate build and run stages

VI. Processes
Execute app as stateless processes

- Backing Services
  - Service consumed by app as part of normal operations
    - DB, Message Queues, SMTP servers
    - May be locally managed or third-party managed
  - Services should be treated as resources
    - Connected to vis URL / Configuration
    - Swappable (change in – memory DB for MySQL)

12- Factor Application

IV. Backing Services
Treat backing services as attached resources

V. Build, Release, Run
Strictly separate build and run stages

VI. Processes
Execute app as stateless processes

- Build, Release, Run
  - Build stage – converts codebase into build (version)
    - Including managed dependencies
  - Release stage – build + config = release
    - Ready to run
  - Run – Runs app in execution environments

12- Factor Application

IV. Backing Services
Treat backing services as attached resources

V. Build, Release, Run
Strictly separate build and run stages

VI. Processes
Execute app as stateless processes

- Processes
  - One or more discrete running processes
  - Stateless
    - Processes should not store internal state (HTTP Sessions)
  - Shared Nothing
    - Data needing to be shared should be persisted
  - Memory / local tmp storage considered volatile
  - Processes may intercommunicate via messaging / persistent storage

12- Factor Application

VII. Port binding
Export services via port binding

VII. Concurrency
Scale out via the process model

IX. Disposability
Maximize robustness with fast startup and graceful shutdown

- Port binding
  - App should not need a “Container”
    - Java App Server, Apache HTTPD for PHP …
    - PaaS now takes that role
  - Apps should export HTTP as a service
    - Define as a dependecncy (#2)

      Tornado (Python) , Thin (Ruby), embedded Jetty/Tomcat (Java)
    - Execute at runtime
  - One App can become another App’s service (#4, #6)

12- Factor Application

VII. Port binding
Export services via port binding

VII. Concurrency
Scale out via the process model

IX. Disposability
Maximize robustness with fast startup and graceful shutdown

- Concurrency
  - Processes are first class citizens
    - Like Unix service daemons
    - Unlike Java threads
  - Individual processes are free to multithread
    - BUT a VM can only get so large (vertical scaling).
    - Must be able to span multiple machines (horizontal scaling)

12- Factor Application

VII. Port binding
Export services via port binding

VII. Concurrency
Scale out via the process model

IX. Disposability
Maximize robustness with fast startup and graceful shutdown

- Disposability
  - Processes should be disposable
    - Remember, they’re stateless!
  - Should be quick to start and stop
    - Should exit gracefully / finish current requests.
    - Or should be idempotent / reentrant
  - Enhances scalability and fault tolerance
  - Design *crash-only* software

12- Factor Application

X. Dev/ prod parity
Keep dev, staging, prod as similar as possible

XI. Logs
Treat logs as event streams

XII. Admin processes
Run admin/ mgmt tasks as one-off processes

- Development, Staging, Production should be similar
  - Dev / prod environments often different
    - Tool gap – dev use SQLLite / Nginx prod uses Apache/ Oracle
    - Personnel gap – developers develop, admins deploy
    - Time gap – (development over weeks / months)
  - Keep differences minor
    - Reduce tool gap – use same software
    - Reduce time gap – small changes & continuous deployment
    - Reduce personnel gap – involve developers in deployment and monitoring

12- Factor Application

X. Dev/ prod parity
Keep dev, staging, prod as similar as possible

XI. Logs
Treat logs as event streams

XII. Admin processes
Run admin/ mgmt tasks as one-off processes

- Logs are streams of aggregated , time –ordered events
  - Apps are not concerned with log management
    - Just write to sysout.
  - Separate log managers handle management
    - Logging as a service
- Can be managed via tools like papertrail, Splunk …
  - Log indexing and analysis

12- Factor Application

X. Dev/ prod parity
Keep dev, staging, prod as similar as possible

XI. Logs
Treat logs as event streams

XII. Admin processes
Run admin/ mgmt tasks as one-off processes

- Admin Processes / Management Tasks Run as One-Off Processes.
  - DB Migrations, one time scripts, etc.
  - Use same environment, tool, language as application processes
    - REPL

Read – Eval – Print Loop = command-shell for running non-interactive shell scripts

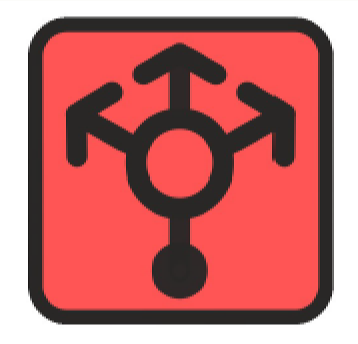
Design Guidelines

- Application Architecture
  - Application architecture concerns:
    - Load Balancing / Session Management
    - Local file system
    - Port Limitations

Load Balancing Router

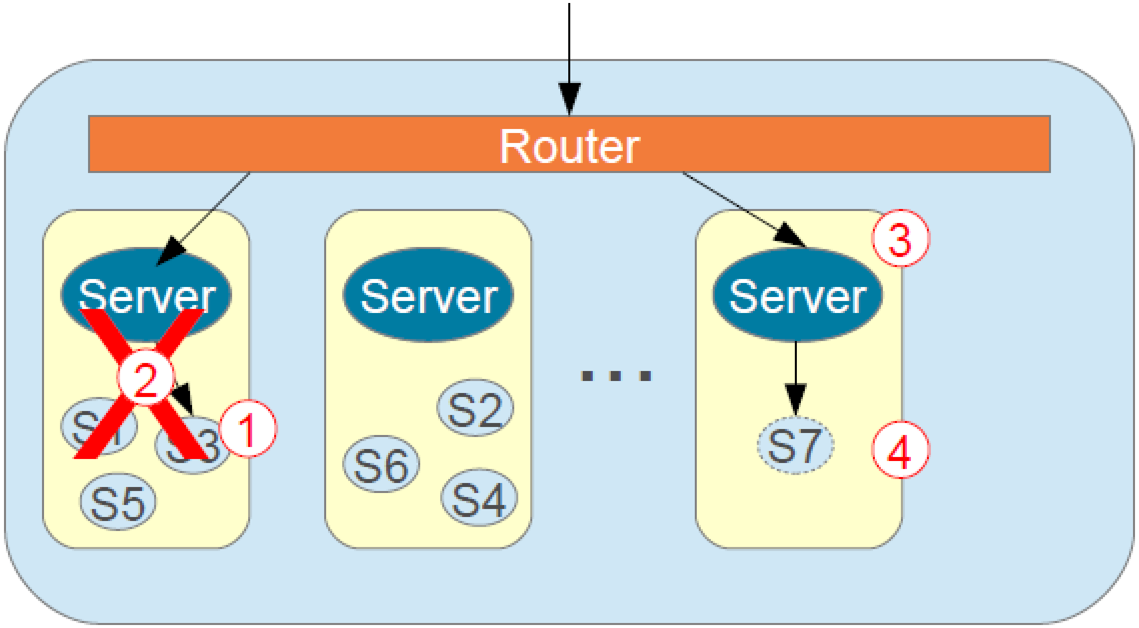
- CF Router provides automatic load balancing
  - When >1 instance
  - Uses HAProxy http://www.haproxy.org

- Sticky Session – based on JSESSIONID parameter
  - Works automatically for Java Web apps
  - Other technologies need extra steps



Session Management

- Based on sticky sessions, managed by Router
- Session is NOT persisted between instances.
  - If an instance fails, those sessions are lost.



1 Requests *stick* to previous session

2 Server dies

3 New container & server started

4 New session – old session lost

Session Management

- Session use best avoided
  - In order to achieve massive scaling
  - Easy for RESTful servers
- If sessions are essential
  - Add persistent session management
    - For example: Gemfire cache
  - Move session-data to light-weight persistent store
    - Such as Redis key-value store

All work described was performed by Capgemini or a Capgemini affiliate

Local File Access

- Apps should not attempt to access the local file system
  - Short lived, not shared
- Instead, use Service abstraction when flat files are needed
  - Amazon S3, Google Cloud Storage, Dropbox or Box
    - Examples: file – uploading
  - File Storage as a Service is coming
- Or consider using database
  - Redis : Persistent, in – memory data
  - Mongo DB : JSON document storage

Logging

- Loggregator will automatically handle all output logged to sysout or syserr

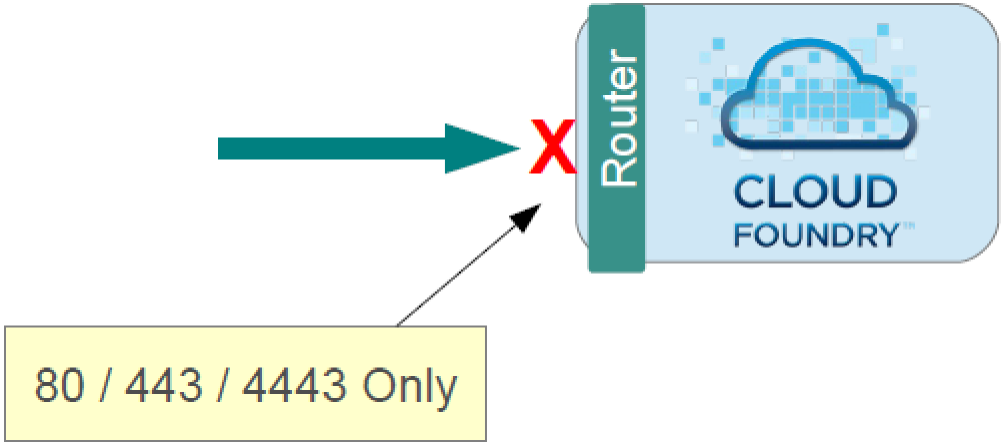
- Don’t use log – files
  - Local file system is generally not available
  - Loggregator will NOT handle files made to the file system or other sources
  - Write to sysout instead
  - Or consider writing log records to a fast, NoSql database
    - Can now be queried

Resources

- All needed resources should be available via classpath
  - Example: use **classpath** : resource in Spring
- File resource not available
  - Short lived / not shared
- Place configuration in classpath: resources
  - Spring MVC supports static web-resources in jars
    - Such as CSS, HTML, images, …

Port Limitations

- Port usage currently limited to HTTP and HTTPS
  - Only 80, 443 and 4443* open to incoming traffic
    - Outgoing traffic controlled by Security Groups
  - Cloud Foundry Router only supports these protocols



* 4443: for secure websockets

Recap

12-Factor

logging

local files

Session management

resource

port

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

People matter, results count.

About Capgemini

With more than 130,000 people in 44 countries, Capgemini is one of the world's foremost providers of consulting, technology and outsourcing services. The Group reported 2012 global revenues of EUR 10.3 billion.
Together with its clients, Capgemini creates and delivers business and technology solutions that fit their needs and drive the results they want. A deeply multicultural organization, Capgemini has developed its own way of working, the Collaborative Business Experience^{TM}, and draws on Rightshore$^{®}$, its worldwide delivery model.

www.capgemini.com

The information contained in this presentation is proprietary.