Designing Applications for Cloud Foundry

Writing Applications that Scale

Getting it right *By Design*

Overview

- After completing this lesson, you should be able to:
  - Consider design considerations for Cloud Foundry

Roadmap

- 12-Factor Applications
- Design Guidelines

Developer Topics / Application Architecture

- Applications may require adjustments to run successfully in Cloud environment
  - See Developer Topics:
  - http://docs.cloudfoundry.org/devguide/deploy-apps/prepare-to-deploy.html

Roadmap

- **12-Factor Applications**
- Design Guidelines

12-Factor Application

- http://12factor.net
- Outlines architectural principles for modern apps
  - Focus on scaling, continuous delivery, portable, and cloud ready

Pivotal™

12-Factor Application

I. Codebase
One codebase tracked in SCM, many deploys

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

VIII. 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

Pivotal™

12-Factor Application

I. Codebase	II. Dependencies	III. Configuration
One codebase tracked in SCM, many deploys	Explicitly declare and isolate dependencies	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	II. Dependencies	III. Configuration
One codebase tracked in SCM, many deploys	Explicitly declare and isolate dependencies	Store config in the environment

- Dependencies
  - Packaged as jars (Java), RubyGems, CPAN (Perl)
  - Declared in a 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 deploys

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 codebase be made open source?
    - Internal wiring (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
    - <u>Connected</u> to via URL / configuration
    - <u>Swappable</u> (change in-memory DB for MySQL)

Pivotal™

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 environment

12-Factor Application

IV. Backing Services	V. Build, Release, Run	VI. Processes
Treat backing services as attached resources	Strictly separate build and run stages	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

Pivotal™

12-Factor Application

VII. Port binding	VIII. Concurrency	IX. Disposability
Export services via port binding	Scale out via the process model	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 dependency (#2)
      - Tornado (Python), Thin (Ruby), embedded Jetty/Tomcat (Java)
    - Execute at runtime
  - One App can become another App's service (#4, #6)

http://www.grahambrooks.com/2014/04/07/container-less-web-applications.html

Pivotal™

12-Factor Application

VII. Port binding

Export services via port binding

VIII. 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	VIII. Concurrency	IX. Disposability
Export services via port binding	Scale out via the process model	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	XI. Logs	XII. Admin processes
Keep dev, staging, prod as similar as possible	Treat logs as event streams	Run admin / mgmt tasks as one-off processes

- Development, Staging, Production should be similar
  - Dev / Prod environments often different
    - Tool gap – devs 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

Pivotal™

12-Factor Application

X. Dev/prod parity	XI. Logs	XII. Admin processes
Keep dev, staging, prod as similar as possible	Treat logs as event streams	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

Pivotal™

12-Factor Application

X. Dev/prod parity	XI. Logs	XII. Admin processes
Keep dev, staging, prod as similar as possible	Treat logs as event streams	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, tools, language as application processes
    - REPL

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

12-Factor Application

I. Codebase One codebase tracked in SCM, many deploys	**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	**VIII. 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

Pivotal™

Roadmap

- 12-Factor Applications
- **Design Guidelines**

Application Architecture

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

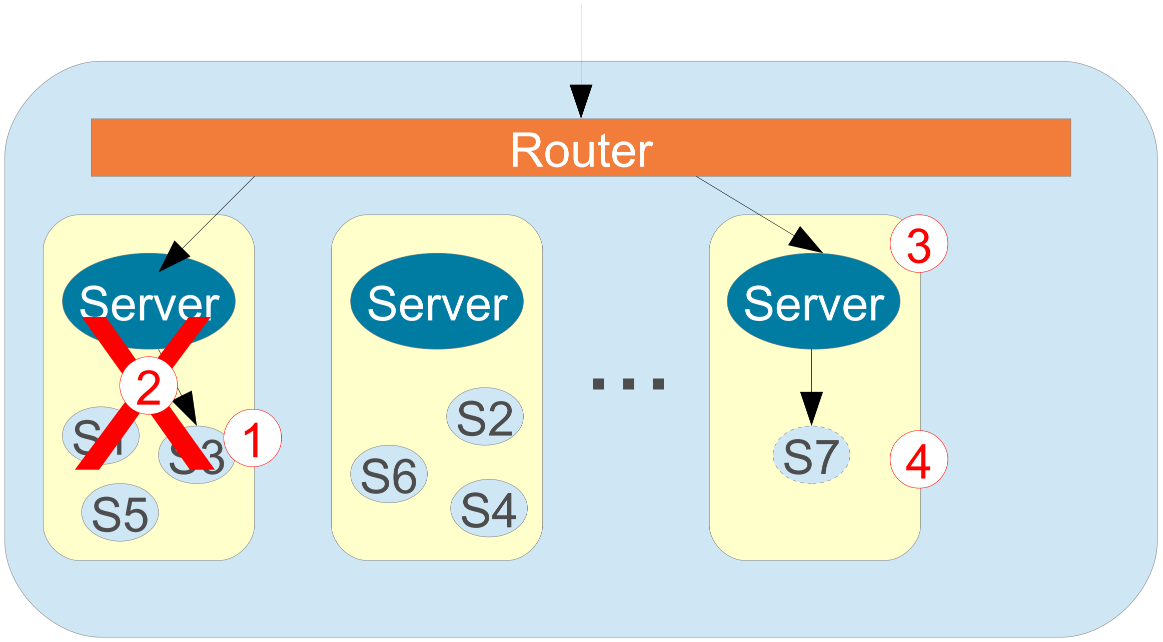
Load Balancing Router

- CF *Router* provides automatic load balancing
  - When > 1 instance

- Sticky Sessions - 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 a light-weight persistent store
    - Such as Redis key-value store

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 a database
  - Redis: Persistent, in-memory data
  - Mongo DB: JSON document storage

Pivotal™

Logging

- Loggregator will automatically handle all output logged to sysout or syserr
  - **Note:** **`cf logs`** *receives* data on port 4443 (typically blocked by corporate firewalls)
- Don't use log-files
  - Local file system is generally not available
  - Loggregator will NOT handle log 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

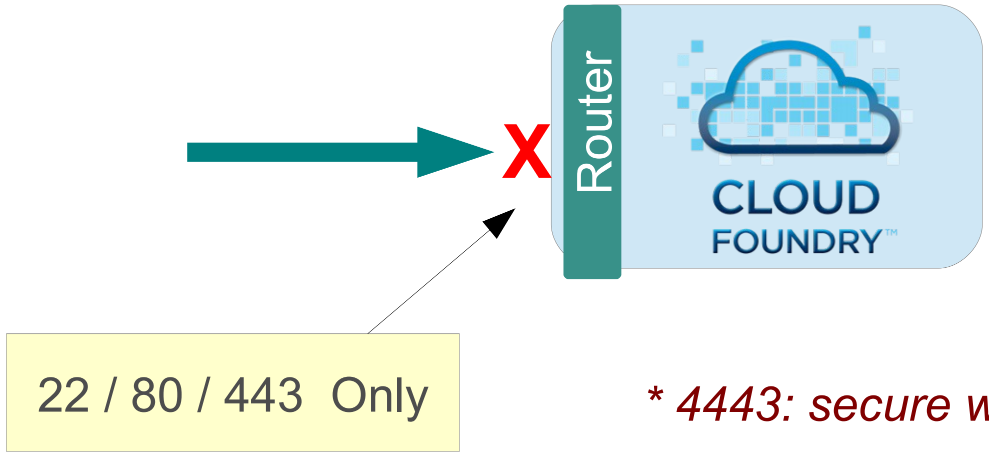
Resources

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

SSH since PCF 1.6

Port Limitations

- Port usage currently limited to SSH, HTTP and HTTPS
  - Only 22, 80, 443 open to *incoming* traffic
    - Outgoing traffic can be controlled by **Security Groups**
    - Open 4443 *inbound* in *your* firewall for logging
  - Cloud Foundry *Router* only supports these protocols



** 4443: secure websocket for PCF logging*

Summary

- After completing this lesson, you should have learned:
  - Architectural design factors for building scalable applications in Cloud Foundry