Pivotal Cloud Foundry®

Cloud Native Design

Includes 12 Factor Apps

Pivotal

Topics

- 12-Factor Applications
- Cloud Native Design Guidelines

Pivotal

12-Factor Application

- http://12factor.net

- Outlines architectural principles and patterns for modern apps
  - Focus on scaling, continuous delivery, portable, and cloud ready
- Most of these principals are built in to the Cloud Foundry platform...

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 codebases = distributed system (not an app)
  - Tracked in version control
    - Git, Subversion, Mercurial, etc.
  - Multiple deployments
    - Development, testing, staging, production, etc.
    - Don't hardcode anything that varies with deployment

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.
  - Don't rely on implicit dependencies from the deployment environment

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

- Configuration
  - Anything that varies by deployment should not be hard-coded
  - Environment variables or configuration server recommended

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

- 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 via URL / configuration
    - Swappable (change in-memory DB for MySQL)

Pivotal

12-Factor Application

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

- Build, Release, Run
  - Build stage – converts codebase into build (version)
    - Including managed dependencies
  - Release stage – build + config = release
  - Run – Runs app in execution environment
- In Cloud Foundry, these stages are clearly separated with **`cf push`**

Pivotal

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
  - The app executes as one or more discrete running processes
  - Stateless
    - Processes should not store internal state
  - Share nothing
    - Data needing to be shared should be persisted
  - Any necessary state is externalized as a backing service

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
  - The app is self-contained
    - For example, Tomcat is included in the droplet
  - Apps are exposed via port binding (including HTTP)
    - Every app instance is accessed via a URI and port number
  - One app can become another app's service

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
  - Achieve concurrency by scaling out horizontally
    - Scale by adding more app instances
  - Individual processes are free to multithread

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
    - Enhances scalability and fault tolerance
  - Should exit gracefully / finish current requests

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
  - This enables high quality, continuous delivery

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 stdout
  - 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

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
  - Run admin processes on the platform
    - Leverages platform knowledge and benefits
  - DB migrations, one time scripts, etc.
  - Use the same environment, tools, language as application processes

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

Topics

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

Pivotal

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

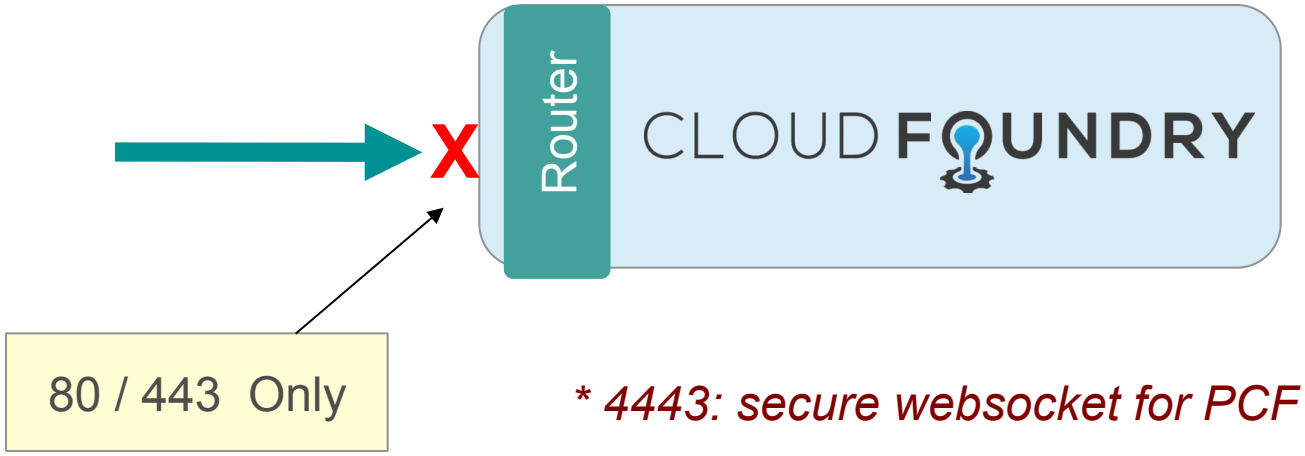
- Apps should not attempt to access the local file system
  - Short lived, not shared
- Instead, use Service abstraction when flat files are needed
  - Amazon S3, Google Cloud Storage, Dropbox, or Box
    - Examples: file-uploading
  - File Storage as a Service is coming
- Or consider using a database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

Logging

- Loggregator will automatically handle all output logged to stdout 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 stdout instead
  - Or consider writing log records to a fast, NoSql database
    - Can now be queried

Port Limitations

- Incoming port usage currently limited to HTTP and HTTPS
  - Only 80, 443 open to *incoming* traffic
- App instances can make calls to any port or protocol and are controlled by Application Security Groups
  - Open 4443 inbound in *your* firewall for logging



** 4443: secure websocket for PCF logging*

Pivotal

Summary

- 12-Factor Applications
  - Designing for *The Cloud*

- Cloud Native Design Guidelines
  - Avoid using sessions or local file-system
  - Log to system error or system out
  - Restricted to HTTP and HTTPS

Pivotal