Updated: Jan 9th, 2017

IBM Bluemix Development & Certification

Summary decks for a course that covers the A to Z of IBM Bluemix.

For more information visit:
http://www.acloudfan.com

raj@acloudfan.com

1. Cloud native apps
2. 12 factors methodology

PS: Certification practice test questions NOT available in the summary decks

Discounted access to the courses:

https://www.udemy.com/ibm-bluemix/?couponCode=BLUE100

Coupon Code = **BLUE100**

https://www.udemy.com/rest-api/?couponCode=REST100

Coupon Code = **REST100**

PS:

- For latest coupons & courses please visit: http://www.acloudfan.com
- Enter to **WIN Free access** – please visit: http://www.acloudfan.com/win-free-access

raj@acloudfan.com
http://www.acloudfan.com

Cloud Native Applications

Cloud Native Application

- Traditional architecture and design practices for application are not aligned with the cloud platform
- Cloud native applications:
  - Built to be self healing (automation & redundancy)
  - Take advantage of the cloud computing platform(s)
  - Scale up or down based on the defined policies
  - Designed for failure

raj@acloudfan.com
http://www.acloudfan.com

12 factor methodology

http://12factor.net/

- Best practices for the development of applications meant to be deployed on a cloud platform
- The 12 factor app is a methodology for building SaaS that:
  - Uses declarative format for setup automation
  - Suitable for deployment on the cloud platform
  - Clean contract with the underlying resources to maximize portability
  - Minimize divergence between production and development environments

raj@acloudfan.com
http://www.acloudfan.com

12 Factors

#1 Codebase

THE TWELVE-FACTOR APP

"One codebase tracked in revision control, many deploys"

- Single codebase for apps in revision control (GIT, Subversion…)
- Each app in its own repository; Branches used for deployments to environments

#2 Dependencies

THE TWELVE-FACTOR APP

"Explicitly declare and manage dependencies"

- Explicitly declare all app dependencies such as Jar files and node JS packages
- Automate the build process – repeatable deployments
  - E.g., node Js applications list dependencies in *package.json*
    npm command takes care of downloading & packaging the dependencies
  - E.g., use Maven for Java/Spring apps.
    1. explicitly declares the dependencies in Maven files
    2 maven builds app by pulling and packaging dependencies in app war file

#3 Configuration

THE TWELVE-FACTOR APP

"Store configuration in the environment"

- Anything that changes from environment to environment
- Do not place configuration information in the code or property files
- Use environment variables for storing config information
  - E.g., User defined environment variables may be used by developer for setting application specific configuration

#4 Backing services

THE TWELVE-FACTOR APP

"Treat backing services as attached services"

- Attached service = App refers to the service by way of a URL that is provided via environment variables.
- Attach using *cf bind* ; preferably by using manifest file
- Swapping the service would not require any code change
  - E.g., Use user defined service to expose an external data source as a *service* for which url is provided via the environment variable

#5 Build, Release & Run

THE TWELVE-FACTOR APP

"Strictly separate the build and run stages"

Build

- Compile code and package
  e.g., maven used to just create the war file
- One build many deploys

Release

- Droplet created by *cf push*
- War/Jar glued with whatever else is needed e.g., JDK for JAVA, Liberty for container , Tomcat

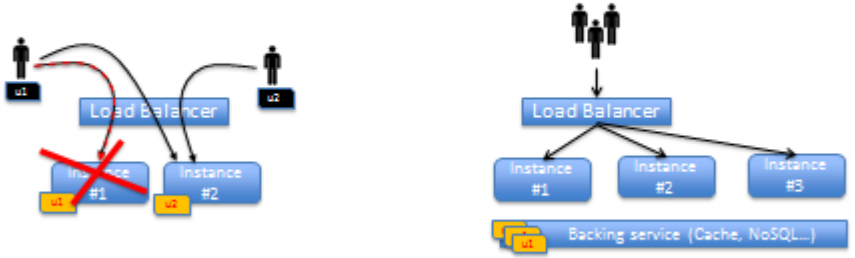
Run

- Run using a single command *cf start*
- Container provides the runtime

#6 Processes

THE TWELVE-FACTOR APP

"Execute the apps as one or more stateless processes"



- No state info in memory or local file system
- State of the system is defined by the data in backing service such as a database

#7 Port Binding

THE TWELVE-FACTOR APP

"Export services via port binding"

- Expose the app like a self contained service with a URL
  - E.g., Java Springboot, Node JS are used to create self contained apps that is they do not need an external web container
- This way one app becomes the backing service for another app

#8 Concurrency

THE TWELVE-FACTOR APP

"Scale out via the process"

- An application process can benefit from vertical scaling up to a certain limit; at some point achieving higher levels of request processing concurrency is not possible
- The application should be designed in such as way that additional process instances may be created to cater to the increased traffic/load

#9 Disposability

"Maximize robustness with fast startup and graceful shutdown"

- Apps may be killed and restarted by cloud platform without notification at any time
- App startup should be fast; as a rule of thumb start up should take no more than 1 minute
- App shutdown should be graceful e.g., close connections, ensure no job is locked

#10 Dev/Prod parity

"Keep Dev, Staging and Production as similar as possible"

- Refers to:
  - Consistent configuration and app setup across all environments
  - Same Backing service (& their versions) across all environments
- Benefits are:
  - Leads to fewer bugs/issues in production
  - Faster delivery & continuous deployment

#11 Logs

"Treat logs as event streams"

- Log to *stdout* and *stderr*
- Cloud platform takes care of
  - Collating the log events/messages from all instances
  - Routing of logs to a destination
- Destination for logs:
  - External to the app; configured by way of *user defined service (cf cups)*
  - Log streams may be analyzed by external systems for generating reports/alerts

#12 Admin Processes

"Run admin/management processes as one off tasks"

- One-off admin processes should be run in an identical environment as the regular processes of the app
  - Build and expose admin processes as part of the app instead of external scripts
- Admin code must ship with the release code so that they are in synch.
  - Developer executing a admin process from his/her laptop is a bad practice