DESIGNING APPLICATIONS FOR CONTAINERIZATION AND THE CLOUD
THE 12 FACTOR APPLICATION MANIFESTO
THIS IS THE DEV PART

DESIGNING OUR APPLICATIONS TO BE PREDICTABLE, FLEXIBLE, RELIABLE, SCALABLE AND COMPLETELY BORING ON A DAY TO DAY BASIS FROM AN OPERATIONAL POINT OF VIEW
THE PUBLIC CLOUD – BEFORE DOCKER

• 2012, Heroku
  • Platform as a Service
  • Small Inexpensive VMs
    • Temporary lifespan
  • Built in load balancing, log aggregation, deployment tooling via git
  • Still had a free tier
  • 12 Factor Manifesto
    • “The contributors to this document have been directly involved in the development and deployment of hundreds of apps, and indirectly witnessed the development, operation, and scaling of hundreds of thousands of apps via our work on the Heroku platform.”
THE TWELVE-FACTOR APPLICATION MANIFESTO
12FACTOR.NET

I. Codebase
   One codebase tracked in revision control, many deploys

II. Dependencies
   Explicitly declare and isolate dependencies

III. Config
   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 the app as one or more 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 development, staging, and production as similar as possible

118x118ilo82x118.jpg

XI. Logs
   Treat logs as event streams

XII. Admin processes
   Run admin/management tasks as one-off processes
• Immutable, self-contained deployment artifact.
• Can run in different operational environments without changes.
• Well instrumented, logically transparent
• Secure
• Predictable
• Easy to scale to different workflow needs
• Easy to develop over time, especially for a team
I. CODEBASE

ONE CODEBASE TRACKED IN REVISION CONTROL, MANY DEPLOYS

- Revision control is not a backup
- Repeatability and the ability to roll back
- What Changed?
- Do Computer Science. Track your experiments.
- Multiple programmers?
- Without it, Docker just makes things worse
  - Version your Docker images, too
• Learn how to branch, review, merge
  • These are basic trades-skills
• Code is conversation
  • With your colleagues
  • With the future
• Organization Matters
• Just use Git. Everyone uses Git
• Automated deployment demands automated testing
II. DEPENDENCIES
EXPLICITLY DECLARE AND ISOLATE DEPENDENCIES

• Deployment artifacts are like space capsules
• Bring it all with you, isolate the environment
• False economy of shared libraries
  • Disk space is cheaper than time
• Fat deployment artifacts
• Build the environment on demand with Puppet
• Bring the server with you via Docker
II. DEPENDENCIES

EXPLICITLY DECLARE AND ISOLATE DEPENDENCIES

- Deployment artifacts are like space capsules
- Bring it all with you, isolate the environment
  - Doesn’t have to be a single file. Just a single directory.
- False economy of shared libraries
  - Disk space is cheaper than time
- Fat deployment artifacts
- Build the environment on demand with Puppet
- Bring the server with you via Docker
II. DEPENDENCIES
EXPLICITLY DECLARE AND ISOLATE DEPENDENCIES

• Deployment artifacts are like space capsules
• Bring it all with you, isolate the environment
• False economy of shared libraries
  • Disk space is cheaper than time
• Fat deployment artifacts
• Build the environment on demand with Puppet
• Bring the server with you via Docker
III. CONFIG

STORE CONFIGURATION IN THE ENVIRONMENT

- "Environment" == “where the application runs”
- Configuration == Tier specific
- Use the same delivery artifact on every tier

Test
- testdb.wisc.edu
- logintest.wisc.edu
- Debug: true

QA
- qadb.wisc.edu
- loginqa.wisc.edu
- Debug: true

Production
- proddb.wisc.edu
- login.wisc.edu
- Debug: false
III. CONFIG
STORE CONFIGURATION IN THE ENVIRONMENT

- Externalization strategies
  - Files on each server
  - Container provided
  - Configuration server
  - Environment variables

<table>
<thead>
<tr>
<th>Test</th>
<th>QA</th>
<th>Production</th>
</tr>
</thead>
<tbody>
<tr>
<td>testdb.wisc.edu</td>
<td>qadb.wisc.edu</td>
<td>proddb.wisc.edu</td>
</tr>
<tr>
<td>logintest.wisc.edu</td>
<td>loginqa.wisc.edu</td>
<td>login.wisc.edu</td>
</tr>
<tr>
<td>Debug:true</td>
<td>Debug:true</td>
<td>Debug:false</td>
</tr>
</tbody>
</table>
III. CONFIG
STORE CONFIGURATION IN THE ENVIRONMENT

• Externalization strategies
  • Files on each server
  • Container provided
  • Configuration server
  • Environment variables

• /etc/myapp/config.yml

  server:
    port: 8082
  datasource:
    url: jdbc:myprod.doit.wisc.edu:1880/mydb
  User: dbuser
  password: dbpassword
III. CONFIG
STORE CONFIGURATION IN THE ENVIRONMENT

• Externalization strategies
  • Files on each server
  • **Container provided**
  • Configuration server
  • Environment variables

• **conf/context.xml**

```xml
<Environment name="spring.profiles.active" value="testqa,local-users" type="java.lang.String" override="false"/>

<GlobalNamingResources>
  <Resource name="jdbc/mydatabaseDS" auth="Container"
    type="javax.sql.DataSource"
    maxTotal="100" maxIdle="30" maxWaitMillis="10000"
    username="my_db_user" password="my_db_user_passwd"
    driverClassName="com.mysql.jdbc.Driver"
    url="jdbc:mysql://database.doit.wisc.edu:3306/mydb"/>
</GlobalNamingResources>
```
III. CONFIG
STORE CONFIGURATION IN THE ENVIRONMENT

• Externalization strategies
  • Files on each server
  • Container provided
  • **Configuration server**
  • Environment variables
III. CONFIG
STORE CONFIGURATION IN THE ENVIRONMENT

- Externalization strategies
  - Files on each server
  - Container provided
  - Configuration server
  - Environment variables

${DB_HOSTNAME}
${DB_USER} ${DB_PASS}
IV. BACKING SERVICES
TREAT BACKING SERVICES AS ATTACHED RESOURCES

• Deployment artifacts should be immutable
• Provide a way to declaratively configure which resources are used
  • Databases
  • Filesystem and cloud storage paths
  • Administrative email addresses
  • Hostnames
• Consider what it does, not what it is
V. BUILD, RELEASE, RUN

STRICTLY SEPARATE BUILD AND RUN STAGES

• Never, ever, ever edit your production artifacts!
  • But when you do, make sure you can propagate that change back to your source code repository.
• Ability to roll software back to a previous state
• Deployment artifacts should be immutable
  • Tagged release executables or archives
  • Uniquely tagged and versioned Docker containers
• Configuration needs to be versioned, too
VI. PROCESSES
EXECUTE THE APP AS ONE OR MORE STATELESS PROCESSES

• Store all non-transient state outside the application
  • The “Shopping Cart”
• Use a dedicated cache if necessary
  • Redis
  • Memcached
  • A database
• User Sessions
VII. PORT BINDING
EXPORT SERVICES VIA PORT BINDING

• Application servers are a dependency
  • BYOAS
  • RAM is cheaper than time
• Isolation vs Sharing
  • Practice Good Operational Hygiene
• Multiple instances

Your App
Operations shouldn’t need to know what’s in here

Your Docker Container
Port 8200
VIII. CONCURRENCY
SCALE OUT VIA THE PROCESS MODEL

• You’ll only need this if your project is a success
• More is better than bigger
• Scale out, not up
• Up is limited, out is limitless
• The rest of the factors prepare for this
• Scaling becomes a financial decision, not a technical one
IX. DISPOSABILITY
MAXIMIZE ROBUSTNESS WITH FAST STARTUP AND GRACEFUL SHUTDOWN

• Cattle, not Pets
• Cycle time affects turnaround time
• Less costly the deployment, the more often you can deploy
• Adaptive scaling
• Chaos Monkey and the Simian Army
X. DEV/PROD PARITY
KEEP DEVELOPMENT, STAGING, AND PRODUCTION AS SIMILAR AS POSSIBLE

• Time Gap
  • How long between deployments?

• Personnel Gap
  • How much do you need to know about the app to deploy it? Who or What deploys it?

• Tools Gap
  • “It worked on my machine.”
  • Docker makes this easier
    • docker run -d -p 49160:22 -p 49161:1521 wnameless/oracle-xe-11
    • docker run --name some-mysql -e MYSQL_ROOT_PASSWORD=my-secret-pw -d mysql:8
  • QA must be identical to production
XI. LOGS

TREAT LOGS AS EVENT STREAMS

• Are your application logs full of information or noise?

• Structured logging
  • `<timestamp> <host> <netid> <operation> ...`

• Mapped Diagnostic Context

• Log Aggregation

• "Disks" in the cloud, or inside a container, are usually ephemeral and often read-only
XII. ADMIN PROCESSES
RUN ADMIN/MANAGEMENT TASKS AS ONE-OFF PROCESSES

• Heroku-ism, Ruby-centric
• Highly restricted access to production services
  • May not be able to access the database or other resources locally
• Reconsider whether admin processes are actually needed
• Consider adding protected endpoints to handle these tasks
• Create a separate application for these tasks
XII.01 METRICS
SEND TELEMETRY AS EVENT STREAMS

• Generic metrics are easy, but not as useful
• Create metrics for your own business processes
• What tells you that the application is working?
• Graphite, Grafana, StatsD, Metrics.NET
• Once you start collecting metrics, you’ll want to collect more metrics. This is a sign you’re doing it right.
XII.02 AUTHENTICATION AND AUTHORIZATION
DON’T LEAVE THIS UNTIL LAST

• Plan for security concerns up front
• Role Based Authorization Control
• Use Shibboleth for Authentication
• Use Manifest for Authorization
TAKE THESE IDEAS WITH YOU

• USE REVISION CONTROL!!!1111!!!
• Deploy self-sufficient artifacts
• Abstract external dependencies
• Store all configuration in the environment
• Use the same artifact on every tier
• Keep copies of your releases. Have releases.
• Assume your app could be killed at any time
QUESTIONS?

- Devops@DOIT Office 365 Team
- Application Design Review Brownbags
  - Every other Friday at Noon, DoIT 3139AB
  - applicationdesignreview@office365.wisc.edu
- Brian Hill <brian.hill@wisc.edu>