Release Engineering II

Idaho State University | Computer Science

Isaac Griffith

CS 2263
Department of Informatics and Computer Science
Idaho State University

ROAR

Outcomes

After today's lecture you will be able to:

- Understand and describe techniques to automate deployment and testing
- Understand how to adopt release engineering
- Describe the types of tools needed and available for automated release engineering

Inspiration

"The process of preparing programs for a digital computer is especially attractive, not only because it can be economically and scientifically rewarding, but also because it can be an aesthetic experience much like composing poetry or music." – Donald Knuth

Aim of Deployment Pipeline

- Every Step should be visible (Results in better collaboration)
- Deliver useful, working software to the users as early as possible
- Early feedback (Identify & resolve issues in the early stages).
- Enable teams to deploy & release any version of their software at any point to any of the environments
- Avoid last day heroics on the release day.
- Release in small chunks (Avoid “Big Bang” release).
- Should be driven by tools & not by individuals.
- Ability for every change to be transparent & propagate them through the pipeline
- Ability to role-back to a previous stable state in case of any failures

Anti-Patterns of Deployment Pipeline

- Manual Deployment of Software
- Deployment performed by multiple teams
- The order of steps are not defined
- No proper “Run books” for failures
- Too much of reliance of manual testing
- Correction to the release process during the actual production release.
- Configuration across all the environments varies to a large extent
- Manual Configuration management of Production Environment
- Deployment to a production-like environment is done only development is 100% complete

Best qualities of a Deployment Pipeline

- Deployments should be automated.
- Deployments should be done frequently (If possible, for every single check-in)
- Provide early feedback so that the development team can act on the failures in a faster way.
- Good Automation Coverage to test early.
- Should be scalable
- Should enable one-click deployment

Large Scale Production Deployment

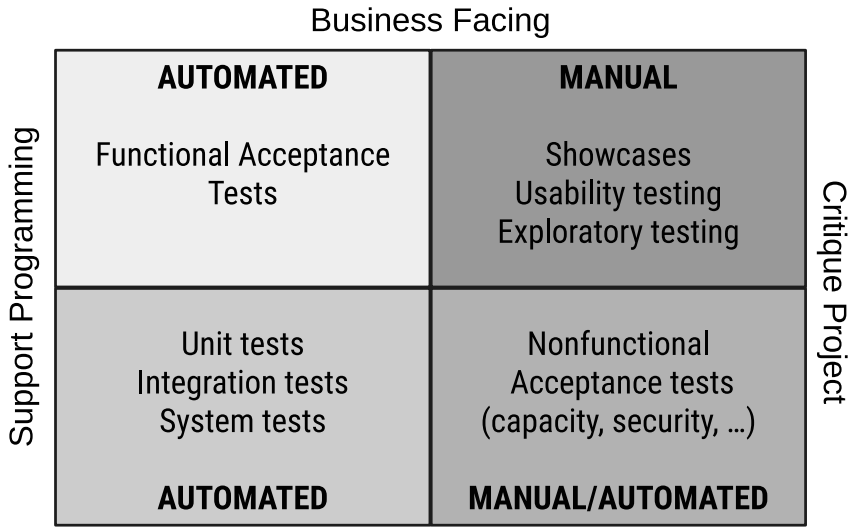
- What are host-type?
- What are recipes / cookbooks?
- Tools like Pogo OR other agent based deployment tools

```
$ ssh <target-host-name> 'whoami;'
```

Why Automated Deployment

- On most occasions, the deployment documentation is outdated
- Logging all the steps in the form of scripts is very beneficial for book keeping purposes.
- Works seamlessly if all the steps are taken care of
- Even non-experts should be able to deploy effortlessly
- Every team using automated deployment scripts results in its maturity & less prone to errors
- Take off the dependency from the deployment expert and utilize them for more challenging tasks.
- Its fast, cheap. Facilitates in early testing & faster feedback cycles.
- Deployment logs are auditable for future references.

Deep dive into Testing Quadrant



ROAR

Automated Acceptance Tests

- Aim
  - Avoid manual tests to whatever extend possible
- Steps to be followed:
  - Tests to be performed in production-like environment
  - If setting up environment is expensive, use a scaled-down environment.
  - Setup the actual user's environment on a grid.
  - Use Business language in the scripts instead of technical jargon ('Place Order' instead of 'Clicking on button XYZ')
  - Well-defined exit criteria for Pass/Fail of tests
  - Don't fail the test due to minor issues

Automated Acceptance Tests cont'd...

- Steps to be followed:
  - Include a few tests to assert external systems
  - If you don't care about a particular field, don't bother testing it.
  - These tests should be run after every deployment
  - Block the pipeline when there are massive failures
  - Parallelize the tests to whatever extent possible
- Outcome of this step is the software packages that have fought against all the tests & challenges like a mythical hero and are ready to take on the world!!!

Few more Generic Anti-Patterns

- Testing done on developer machines
- Service Engineering team hasn't even seen the application till the actual release date
- Installers, Configurations etc…not done till the release date
- No collaboration between development team & SE teams
- Late setup of Staging Environment
- Release too many changes, unable to figure out what caused the failure
- Changing the configurations directly on Production often resulting in disasters

Adoption of Release Engineering

- Change in the mindset of the team members
- All aspects of development, testing, staging, production environments (most importantly configuration management) to be managed through a VCS.
- Integrate development, testing, release teams as a part of the development process
- Manage the infrastructure to build environments in no time
- Beef up the test automation coverage so that there's not much reliance on Manual testing
- Rehearse the release & rollback process so that its easy to do it time & again
- Reach a stage wherein "Software Release" is in self-service mode

Tools Used at each of the stages

- Version Control – Git
- Build Tools – Ant, NAnt, MSBuild, GMake, Mave, Buildr, PSake, Gradle
- Agile / Scrum – Jira
- Static Validation – Coverity, SonarQube
- Unit Testing - JUnit
- Test Automation – Protractor, Selenium
- Continuous Integration – Jenkins, Atlassian Bamboo, Thoughtworks Go, Urban Code AntHillPro
- Deployment – Pogo, Chef
- Environment Provisioning – Puppet, Chef
- IAAS, PAAS – AWS, Azure, Docker, Vagrant

Managing Environments

- No Application is an Island
- Poor Configuration Management results in significant waste of time, additional costs, tech debt, etc…
- Should always be cheaper to create a new environment than repairing a broken environment
- Few of the names used for Environments
  - Development
  - Integration
  - QA/Testing
  - Staging/Pre-production
  - Performance
  - Canary/Bucket Testing
  - Production

Managing Releases of Complex Systems

Orchestration: - Automated arrangement, coordination, and management of complex computer systems, middleware, and services - Aligning the business request with the applications, data, and infrastructure

- Deployment rhythm
- Its about getting the perfect configuration, checking in these values and automate the release using these values
- Have the right sequence of steps in the configuration file
- Have the right kind of checks & balances at every stage
- Practice roll-backs in case of issues
- Implement fail-safe methodologies in case of issues
- Build environments wherein the Production traffic can be replayed (Use of access logs to simulate user actions)

Steps to build a deployment pipeline

- Model your value stream & create a walking skeleton
- Automate the build & deployment process
- Automate unit tests & code analysis
- Automate Acceptance Tests
- Automate Releases

Multi-Tenant Applications

- Customers share the same cloud platform and infrastructure and their data is commingled
- Single code-base for the application
- Upgrades are executed easily by the SaaS provider
- All the tenants are using the same version
- Data of different tenants is stored in the same place, however, measures taken to make it inaccessible to other tenants.
- Bucketize the access based on the IP Range / Cookie range / Header Information etc…
- Toggle a Feature using configuration
- Hard to maintain different versions for different tenants

Multi-Instance Application

- Multiple customers run their own separate instance of an application and OS running on a separate VM.
- Avoid co-mingling of data
- Develop & use their won logical piece of the cloud service
- Allows for greater flexibility and control of configuration, customization, updates and upgrades
- Ability to migrate instance to an on-premise server, or to another cloud hosting provider

Zero Downtime Deployment

- What is ‘Zero Downtime Deployment’?
- What is ‘Out of Rotation’ (OOR)?
- What is ‘Concurrency’?
- Colo (Data-Center) based deployments

Extracting the Server Logs

- Standardize the logging messages. Involves lots of effort from the development teams
- Setup Splunk alerts for ‘FATAL’ errors in the code

Next Generation Tools in the DevOps world

- Chef
- Puppet
- Cfengine
- Salt

Preparation for the Release

- Any issues during the release, stop or postpone the release. Stability takes the highest priority
- Prepare Release Plan or Runbook with correct sets and if possible automate them
- Identify the error-prone steps and automate them
- Perform the drill for performing rolling back of the releases
- Should be as simple as selecting a value and clicking a button
- Let one team perform all the releases. (Too many cooks spoil the broth)

Are there any questions?