# Just Because You Can, Doesn't Mean You SHOULD!

Software Testing in a Micro-Service Architecture

Magenic // Fast Forward

1

# BILL ROSKE

Principal Quality Engineer
Magenic Technologies, Inc.

Email: billros@magenic.com
Twitter: @wroske
Blog: https://qedevangelist.wordpress.com/
https://magenic.com/thinking

Magenic // Software to move forward.™

2

2

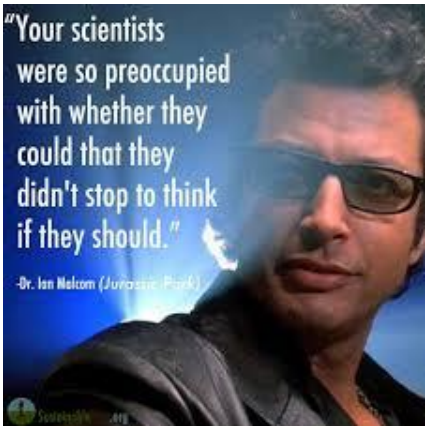
# Intro

- Anyone work at one of these?:
  - Netflix
  - Spotify
  - Facebook
  - Google
  - Amazon
  - Pandora
- Industries?
- How Risk-averse is your Industry?  Company?



# Micro Service Advantages

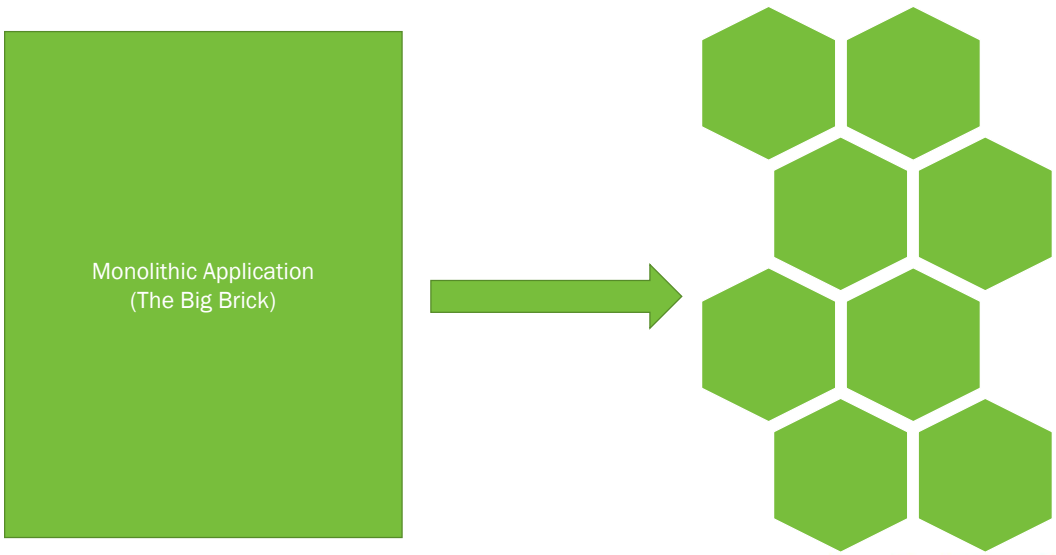
- Speed of Build/Deploy
- Separation of Responsibility
- Lower Complexity
- “Hot Interchange”
- “State-less” – In Theory…
- “Independent” – Sort of…
- Enable Continuous Integration
  - “Role in” a new version
  - Change/Build/Deploy/Test




5

# Micro Service Architecture






6

# Chaos

- Continuous Integration (CI) Vs. Release Control
  - Every Check-in Causes a Build
  - Every Build is Deployed
  - Every Deployment is Tested
- What (How?) Do You Test?



# Netflix Service Map

# Netflix Solution…

THIS IS PRODUCTION!

- Every Service is “Independent”
  - API Versioning
- Deploy: Anyone, Any Time
- Any (ALL) Code in a Service Can Go to Production
  - One Branch
- Feature Flags
  - Limit which/if users get new capabilities
  - Protects incomplete code
- Experiments
  - Try it out on a subset of users
  - How does it perform
  - How many complaints – Yes, even social media



# Netflix Business Model == Yours?

- Dave Hahn – Netflix
  - Message:
    - “Netflix: cannot reach server, try again…”
    - OK or Cancel
  - Alternative:
    - “Go outside and play”
    - SUNSHINE!

- “Nobody ACTUALLY died… We Checked!”

# Challenges for the Rest of Us!

- Client Data Protection
- Constant Change
- Quality Reporting
  - Root Cause Analysis
  - Versioning
- Discover/Audit
- Features Flags
  - Regression for Release
  - "Unreleased" Features?



# Solution

- Feature Flags
- Repositories and Branches
- Create Release Sets
- Utilize Environments

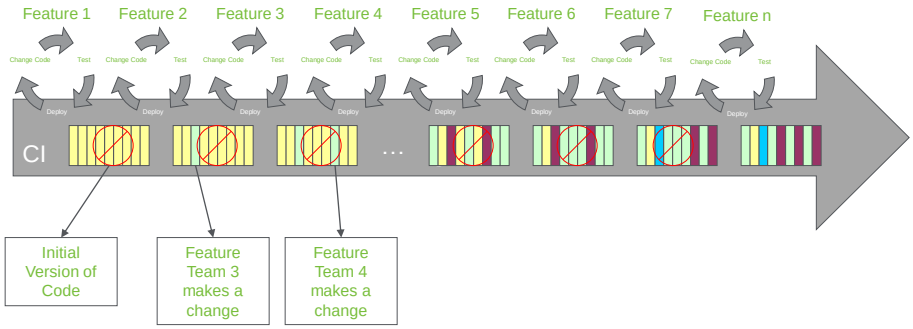
# Feature Flags

- Any Code NOT intended for ALL PRODUCTION users
  - Should be “behind” Feature Flag
  - Default should be OFF for all users
- Additive Features
  - Will not affect existing scenarios or tests.
  - Can be turned on for existing Test Users
- Alternative Features
  - Turn on ONLY for “Feature Test” users, NOT your regression users
- Architectural Features
  - Fundamental change that is NOT flag-able: new Angular Version…
  - Plan Wisely!
  - There is no Fallback



# Repository and Branch Considerations

- 2 Repository options
  - All Services in same Repository
  - Repository per service
- Know and Leverage you Branching Strategy
  - Git Flow
  - Minimal Git Flow
  - Release Branching
- Keep Test Code in the Repository of the Software Under Test
  - Your tests naturally match the Software Version Under Test

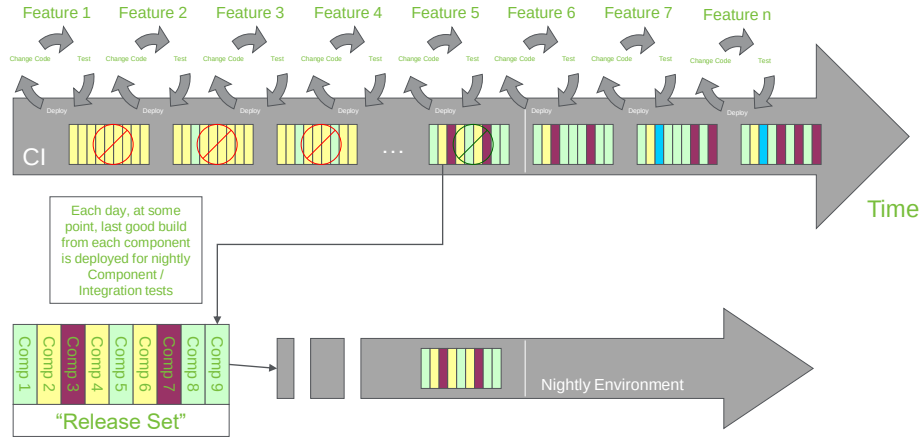



Multiple Feature Teams checking into same component codebase – Continuous Integration



16

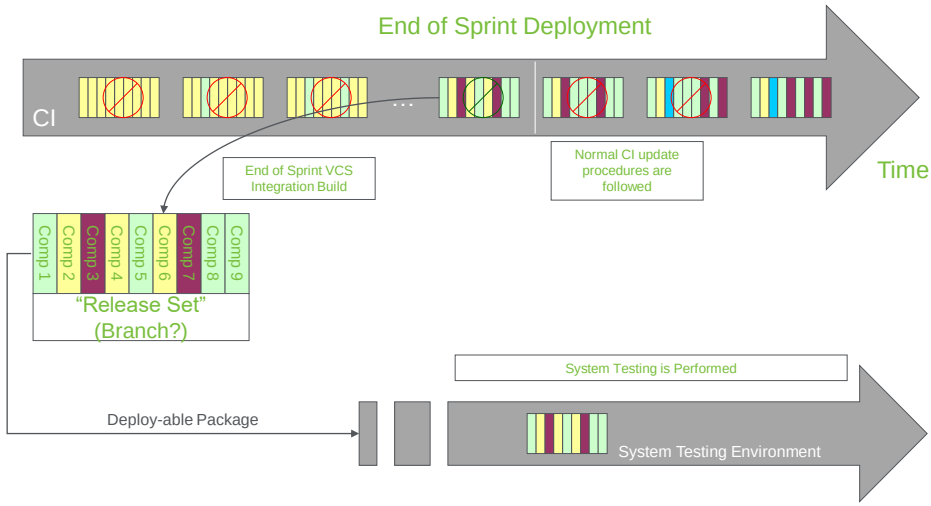
Daily/Nightly Testing




**Overnight Tests:**
- Component Testing
- Integration Testing
- Test Results Report



17






18

# Environments

- Continuous Integration – "Dev"
  - Every Build of Every Service
  - Service Focused Testing
- Test
  - Daily Minimum
- UAT/Performance/Stage
  - End Of Sprint?
- Production

Deploy Release Sets Only



19

# Agility AND Quality in MicroService Architecture

- Automation is Key – Test Often
- DRY-Test: Different Env => Same Code, Different Tests
- Plug into DevOps release pipelines
- Repo's and Branches
- Use Feature Flags
  - Segregate Feature Regression from RC Regression
- Define Release Sets
  - Release Candidates
  - Branch Strategy for Hot-Fixes
- Define Once, Deploy Everywhere
- Roll-Forward



# Questions?


22