Semantic Versioning and Automated Build Assembly

Brian Davenport
John Mallinger
Duane Spence

Approved for Public Release
Build Assembly Introduction

- **DevOps Paradigm**: Build once and deploy many times - don’t rebuild software that hasn’t changed

- **Semantic Versioning**: Identification strategy that indicates type of changes and build compatibility included in a build by using defined rules and naming convention

- **System Build**: Assembled set of compatible component and configuration item builds along with products necessary to support install, deploy, and configuration.
Semantic Versioning: An Overview

- Build Once - do not rebuild if nothing has changed
- Version identifier drivers indicator of types of changes in a build
  - Major version: change breaks compatibility with external consumers
  - Minor version: new capabilities delivered, but change is compatible to external consumers
  - Patch version: updates and discrepancy fixes in existing capabilities

M.m.p (e.g. 2.3.0)

1) If Developer introduces an incompatible change, they update Major version # & reset the Minor & Patch version #s (M+1).0.0 (e.g. 3.0.0)
2) If a patch is required to previously released version, the developer increments Patch version # M.m.(p+1) (e.g. 2.3.1)
3) Version is same across all products built at the same time (e.g. EIS sessions all have same version)

Approved for Public Release
Drivers for System Assembly

- The problem: Drowning in DevOps builds
  - 8 Configuration Items
  - 160 Custom Software Components
  - 800+ builds per week
  - 200+ COTS/FOSS products

- After we build and test individual components, how do we assemble into a system?

- A component changed - what do we need to build and retest?
System Assembly Pipeline

- Automated selection of component builds
  - Incorporates outputs of component pipelines
  - Test results tracking from component pipelines
  - Compatibility Validation
  - Tailoring build outputs for target deploy environments

- Automated Deploy & Checkout
  - Stage and deploy system build into representative environment
  - Run integration thread tests to validate compatibility and core functionality
  - Tag system build with test results and assess viability for promotion to ops
System View

System Build

Mission Build
- CI1
- CI2
- CI3
- CI4
- CI5
- CI6

Middleware CI

Platform Build
- Storage
- Network
- Compute

Simulator Build
- CI7
- CI8

Simulation Framework

Approved for Public Release
Layered Pipelines Driving CI Checkout

Platform Pipeline

Build / Assemble → Unit Test → Deploy → Audit → Functional Test → Security Test → VM Image

Middleware Pipeline

Build → Unit Test → Deploy → Audit → Functional Test → Security Test → VM Image

Mission Pipeline

Build → Unit Test → Deploy → Audit → Functional Test → Security Test → Artifact Repository

System Assembly

Assemble → Stage → Deploy → Functional Test → Artifact Repository

Approved for Public Release
System Assembly Pipeline Overview

Assemble
- Create Assembly Template & Ruleset Template
- Run Compatibility Algorithm
- Publish System Compatibility Matrix

Stage
- Download and Stage Artifacts

Deploy
- Deploy Platform, Mission, and Simulator

Functional Test
- Run Functional Tests against deployed env.
- Update test results

Artifact Repository

Approved for Public Release
System Release: Assembly

Assembly Template:
1. Lists all of the files contained in a release
2. Identifies files and tags to component, CI, and build

Rule Set Template:
1. Defines checks / parameters to validate when retrieving files from Artifactory repository
2. References and pulls in data from Compatibility Input Table for rule execution and enforcement

Compatibility Input Table
1. Defines major changes and impacted components
2. Supports tracking component change alignment

Artifact Repository

Approved for Public Release
## Case Study: Compatibility Matrix

### Release Managers

- **Release Planning Meetings**

### Release Compatibility Matrix

List only those Changes that impact major # of others

<table>
<thead>
<tr>
<th>Change#</th>
<th>Status</th>
<th>CI1</th>
<th>CI2</th>
<th>CI3</th>
<th>CI4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Change1</td>
<td>In-Progress</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Change2</td>
<td>In-Progress</td>
<td>4</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>Change3</td>
<td>In-Progress</td>
<td>5</td>
<td>3</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>Change4</td>
<td>In-Progress</td>
<td>6</td>
<td>4</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>Change5</td>
<td>In-Progress</td>
<td></td>
<td></td>
<td>4</td>
<td>6</td>
</tr>
</tbody>
</table>

### Artifactory Component Versions

<table>
<thead>
<tr>
<th>Date</th>
<th>CI1</th>
<th>CI2</th>
<th>CI3</th>
<th>CI4</th>
</tr>
</thead>
<tbody>
<tr>
<td>4/16</td>
<td>2.5.0</td>
<td>1.2.0</td>
<td>1.0.0</td>
<td>1.5.0</td>
</tr>
<tr>
<td>4/18</td>
<td>3.0.0</td>
<td>2.3.0</td>
<td>1.0.1</td>
<td>1.6.0</td>
</tr>
<tr>
<td>4/19</td>
<td>4.0.0</td>
<td>2.4.0</td>
<td>1.1.0</td>
<td>2.0.0</td>
</tr>
<tr>
<td>4/20</td>
<td>4.0.1</td>
<td>2.4.1</td>
<td>2.0.0</td>
<td>3.0.0</td>
</tr>
<tr>
<td>4/25</td>
<td>4.0.2</td>
<td>2.4.2</td>
<td>2.1.0</td>
<td>3.1.0</td>
</tr>
<tr>
<td>4/30</td>
<td>4.1.0</td>
<td>2.4.3</td>
<td>2.2.0</td>
<td>4.0.0</td>
</tr>
<tr>
<td>5/1</td>
<td>5.0.0</td>
<td>2.4.4</td>
<td>2.3.1</td>
<td>4.1.0</td>
</tr>
</tbody>
</table>

### Release Manifest File VersionContent

<table>
<thead>
<tr>
<th>Job's Run Date &gt;&gt;</th>
<th>4/18</th>
<th>4/19</th>
<th>4/22</th>
<th>4/30</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rel# Mission_1.0</td>
<td>1.0.0.23</td>
<td>1.1.0.34</td>
<td>2.0.0.43</td>
<td>2.1.0.53</td>
</tr>
</tbody>
</table>

**CM Repo.**

- CI1 interface change impacts CIs 2, 3, and 4

**Updates delivered and complete pipeline at different times**

**Change injected when all CIs deliver & pass pipeline checkout**

---

Approved for Public Release
Designing for System Assembly

- Identify lowest level at which a system can / should be patched
  - Structure versioning strategy and build strategy to be independent at that level
  - Needs to account for patching / update CONOPS and available outages

- Define dependencies associated with each component and CI
  - Need to capture both interfaces and APIs / shared libraries
  - Also track runtime and

- Decompose build assembly architecture
  - Strategy for rolling up and combining system builds
  - Approach for validating compatibility across components and subsystems

- Assess core set of tests needed to promote build
  - What component / CI checkouts are required as part of each lower level build?
  - What system-level tests should be completed to validate system build integration?
  - What is an acceptable risk / discovery posture for the next downstream user?
Lessons Learned

- Drive culture that supports continuous change delivery
  - Continuous deliveries of change
  - Always maintain a working build and working system
  - Requires more product oriented mindset

- Legacy CM strategies limited build and versioning at component level
  - Goal: Component level builds and patches for every mission software component
  - Issue: Extensive cross-component dependencies within a CI drove some cases of larger builds

- Design automated tests for reuse across all I&T environments
  - Automated tests can spread out beyond their initial purpose
  - Leverage common test architecture across all environments to maximize reuse and avoid refactoring