adaptTo()

EUROPE'S LEADING AEM DEVELOPER CONFERENCE

27th – 29th SEPTEMBER 2021

AEMaaCS Cloud Manager Build Deciphered

Konrad Windszus, netcentric

netcentric
A Cognizant Digital Business

About Me

- Konrad Windszus, konrad@netcentric.biz
- Passionate about Open Source
  - Apache Software Foundation member
  - PMC member of Apache Sling and Apache Jackrabbit
  - ACS AEM Commons committer
- Working at Netcentric

Polls

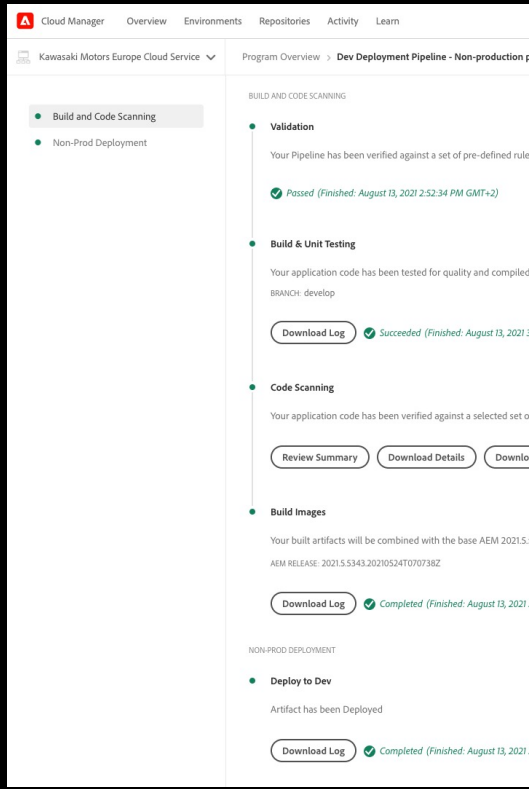
Warning

- All statements are mine and based on observation as documentation for the Cloud Manager build is sparse
- Take everything with a grain of salt
- *Things may change any time*

Cloud Manager Build Phases & Steps

- Important Steps
  1. Build and Unit-Testing
  2. Build Images
  3. Deploy to Dev / Stage / Production

- Phases named/aggregated differently in Dev and Prod pipelines

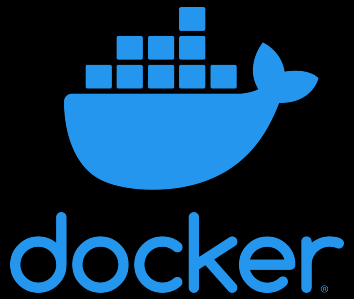


1. Build and Unit Testing

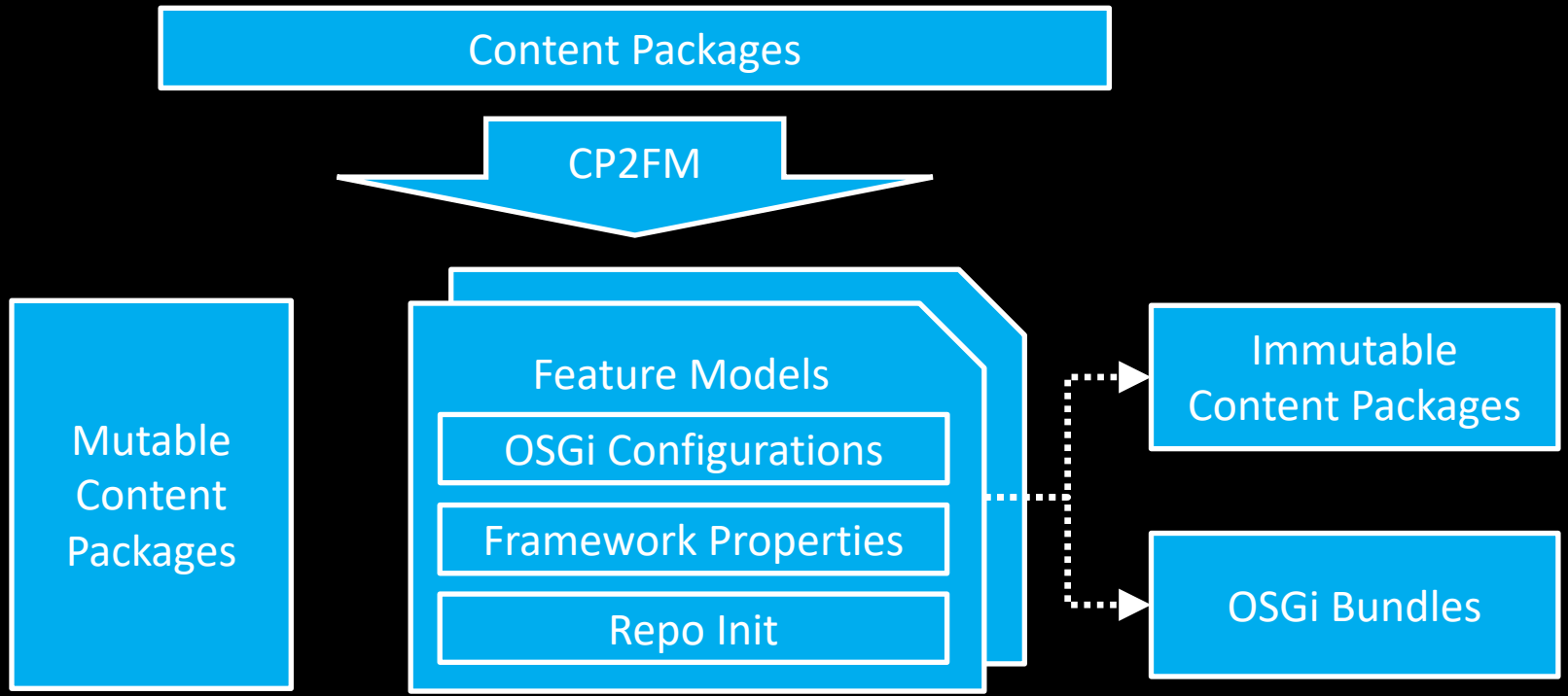
- Regular Maven Build
  - Phase: `package`
  - Ubuntu, Maven 3.6.0, configurable JDK (default: Oracle 8)
- Best practices:
  - Define explicit Java version (`.cloudmanager/java-version`)
  - Mark only one content package container for deployment
  - For aggregating different tenants use *git submodules*
  - Prevent using native binaries (e.g. in NPM build)

2. Build Images

1. Conversion of content packages
2. Docker build process
    1. AEM Author
    2. AEM Publish
    3. AEM Publish Dispatcher



2. Build Images
Conversion of content packages



2. Build Images
Conversion of content packages

Verbose and Default Bundle Start Order

For merging repoinit

cp2fm be ex ally

- CLI a guments vi Manager Log

```
cp2fm -v -b 20 --seed-feature=file:/root/.m2/repository/com/adobe/granite/aem-
eth                                                    ma
addons-auth        osgifeature
-a /root/.m2/repository/ -o src/main/fea
-i '${project.groupId}:${project.artifactId}:
-e global -r com.adobe.aem.deprecated
-f '.*/(apps|libs)/(.*)/install\.(((author|publish)\.(dev|stage|prod))|
((dev|stage|prod)\.(author|publish))|(dev|stage|prod))/(.*)(?<=\.(zip|jar)$)'
/tmp/packages/digitalxn-aem-all-container-1.18.0-SNAPSHOT.zip
```

Artifacts and features output directory

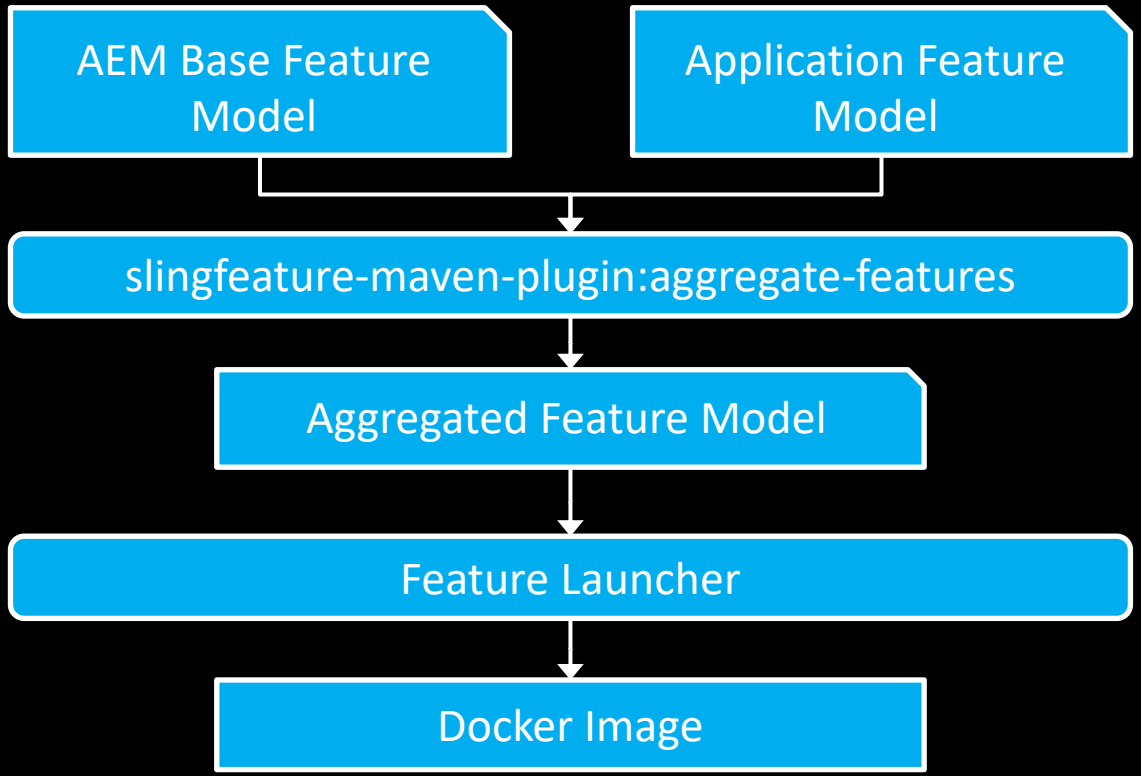
Generated feature artifact id

API region used for exports and feature

Disallowed package entries

Input Content Package

2. Build Images
Creation of AEM Docker Images



2. Build Images
Prepare AEM Docker Image

1. Start AEM
    1. Install all packages from FS package registry
    2. Precompile Client libraries and scripts
    3. Stop AEM
2. Prepare for Composite Node Store
    1. Remove FS package registry folder
    2. Convert `nt:resource` to `oak:Resource` nodes (with oak-run)
    3. Run compaction
3. Commit Docker image
4. Push to (private) Docker repo

2. Build Images
Dispatcher Docker Image

- Only for Publish
- Merges configuration from Maven build with base configuration
- Dispatcher SDK (with validator) should be used locally

2. Build Images Pitfalls

1. Only volatile mutable content available
   - Mutable Content Packages not installed
2. Only Pipeline Variables available (no Environment Variables)
3. Environment run modes only supported for OSGi configurations (the order matters) not for OSGi bundles and sub packages

3. Deploy

1. Index Update
2. Start Kubernetes AEM Pods
   - Connected to persistent mutable repository
3. Mutable Content Installation
   - Dedicated pod
   - Downloads container package, extract mutable ones
   - Deploys via Package Manager ReST API
     - Deployment to Publish via Replication

3. Deploy
Index Update

- Using Oak API to directly modify (otherwise immutable) segment store and document store
- Creates new version of index definition and index
- Purges indices < current version - 1

3. Deploy Pitfalls

1. AEM Updates not always happening automatically on Stage/Prod
    1. Pipeline Environment variables prevent automatic updates
    2. Update process not (yet?) visible to customer
2. Mutable Packages
    1. Install Hooks not supported on publish
    2. Some repository paths on publish cannot be modified via packages (/var, /tmp)
3. Limitations on custom index definitions

Recommendations

1. Use Maven Tooling for validation
   - aemanalyser-maven-plugin (https://github.com/adobe/aemanalyser-maven-plugin)
   - aem-cloud-validator (https://github.com/Netcentric/aem-cloud-validator)
2. Try to catch failures early
   1. Locally (use Maven tooling, integrate dispatcher validation)
   2. Customer CI (same OS, Java & Maven version as Cloud Manager)
3. Integrate Cloud Manager result into Customer CI
   - Jenkins Plugin available

References

- Mostly inline links in previous slides
- Additional references
  1. https://github.com/cqsupport/cloud-manager
  2. https://adapt.to/2019/en/schedule/deep-dive-into-cloud-native-aem-deployments-based-on-kubernetes.html

netcentric
A Cognizant Digital Business

Questions?