ARCHITECH

Digital Product Studio
Architech's strategy to modern dotCMS development

Going headless and other things
Problems Encountered
Separation of Code and Content

- Frontend assets live in the site browser
- Confusing responsibility
- “Code” assets subjected to same handling as other content assets
Velocity Templates

- Unpopular
- Limited debuggability
- Unit testing challenges
Complex Custom Tooling

- Task runners for processing and deploying VTLs, CSS, and Javascript through WebDAV can be time consuming to develop and maintain
- CI/CD scripts such as Jenkinsfiles and Gradle to deploy static and dynamic plugins have to be customized to deploy all of the artifact types to dotCMS
- Upgrade automation is a challenge
Disruptive of a normal development/deployment workflow

- Changes done through the UI can’t be gated and reviewed in a Pull Request
- Extra work re-producing local work in shared environments
- Long guides to recreate a local environment
- Push publishing for “Application” deployment
Plugins Hurdles

- OSGi is a science
- Upgrading plugins for new dotCMS versions can be delicate
Ideal State for dotCMS development

- Clear separation of code and content
- Modern web development without specialized adaptations
- The state of the solution in dotCMS can be deployed automatically and repeatedly
- Changes are subject to code reviews and gating
- Building and deploying is standard and simple
- Upgrades are easier
Go Headless

Strategy #1
What is “Headless”

- Removing the responsibility of rendering HTML from the CMS
- Implement the web layer using first class web tools
Why go “Headless”

- Remove the Velocity Templating lock-in
- Opens up options to go with “thick client” apps
- Application is a separately deployable package
- Provides a clear separation of code from content for your front-end
- Supports best of breed, highly productive development/deployment workflow
Advantages of a Single Page Application

- A single language to render your application
- Faster site navigation
- Server work offloaded to browser
- Currently there’s only three top choices for SPAs: Angular, React, and Vue
- Clear separation of roles
Options to go “Headless”

- Content as a Service
- Hybrid Layout as a Service
- Pure Layout as a Service
Content as a Service

- Authors only author content instances from defined types
- All of the Information Architecture of the site lives outside of dotCMS
- Simply just retrieving contentlets from dotCMS
- Use an API aggregation layer
Hybrid Layout as a Service

- Authors still have the ability to dynamically layout pages
- Pages still exist in dotCMS’ site browser and are presented in the SPA
- Pages still have templates, templates still have containers, and containers still have rendering Velocity Templating logic
- The SPA has built-in knowledge of the template and knows how to render its layout
- Rendering of the container’s html is still done in dotCMS in Velocity, which is what makes this the hybrid approach
- Currently supported out of the box in dotCMS via the ‘page’ REST api
Pure Layout as a Service

- Similar to Hybrid Layout where pages still exist in dotCMS’ site browser. Page layout is still authorable.
- The SPA has built-in knowledge of specific templates, containers, widgets and how to render them.
- Requires an API that can retrieve the template, containers, and contentlets inside those containers.
Navigation as a Service

- Combine with Layout as a Service
- dotCMS’ `nav` REST API to generate menus
Caveats

- Page Authoring needs to be adapted for LaaS
- Sitemap XML generation may need update
- Sitesearch may need update
- CaaS can circumvent some expected out of the box dotCMS features
Automate Creation of Solution Elements

*Strategy #2*
What gets created through Automation

- Content Types
- Templates
- Containers
- Widgets
- Etc. (Anything a Content Author is not responsible for)
How to Automate

- As a plugin!
- Run on startup
- Use JSON definitions
- Treat it like a migration
- Use predictable Identifiers
Caveats

- Some automation needs may not be supported out of the box
Containerization

Strategy #3
What’s In?

• dotCMS!
• Java JVM
• Dynamic Plugins
• Static Plugins
• Velocity Templates and other static code assets
What’s Out?

- ElasticSearch Indices
- The Assets folder
What does containerization get us?

- A standard build and deploy process
- Simplified CI/CD
- An easy way to get dotCMS up and running
- Easier upgrades
- A fully contained custom dotCMS
- dotCMS no longer an “Application server”
Running Containerized dotCMS
Microservices for Custom Logic

*Strategy #4*
Why separate logic into external Microservices?

- Reduce OSGi and other classloader complications
- Easier tracking, monitoring and isolation
- Choose the right tool for the job
- Meet separate scalability and uptime needs
- Reusable business value not isolated to dotCMS
- Separate features over multiple teams
What to separate into external Microservices?

- Reusable business value
- Meets needs of separate scalability and uptime
- Need for tools that can be a difficult integration into dotCMS
- Can be relatively easy to separate from dotCMS
Conclusion
Strategies to get to the ideal modern state of dotCMS development

• Go Headless
• Automate the creation of fixed structures and mainstays
• Containerize dotCMS
• Externalize customizations into microservices
Q&A