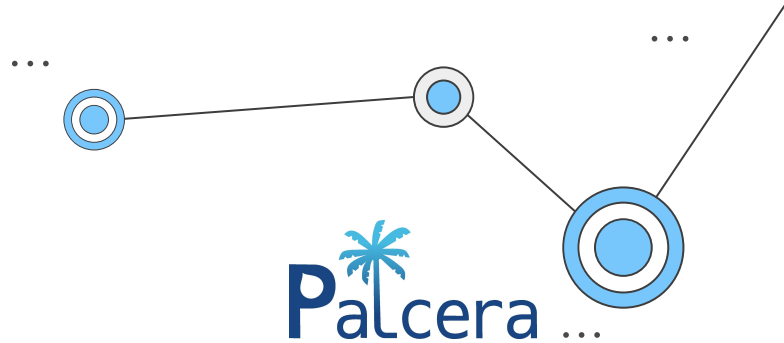
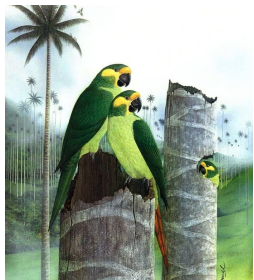




WE'RE SENDING YOU



From Fear to Freedom

Mastering Drupal Updates

Florida Drupal Camp 2025



Carlos Ospina

- Former Technical Account Manager en Acquia
- Drupal Architect Architect
- Palcero www.palcera.com

camoa2005@gmail.com

01

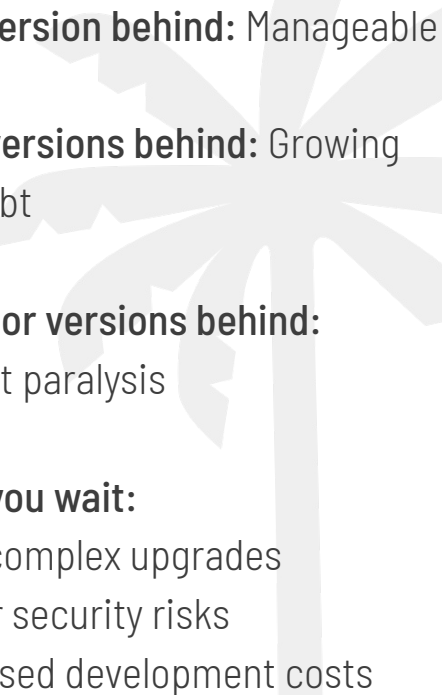
The Cost of Waiting



Technical Debt



The Cost of Waiting



- **One minor version behind:** Manageable
 - **Two minor versions behind:** Growing technical debt
 - **Multiple minor versions behind:** Development paralysis
 - **The longer you wait:**
 - More complex upgrades
 - Higher security risks
 - Increased development costs
- 



02

Systematic Approach

Beyond Random Updates

Updates require structure and planning!

- **Two distinct phases:**
 - Phase 1: Foundation Setup
 - Phase 2: Maintenance Process
- Each phase has specific goals and requirements
- A standard process brings predictability to updates

Laying the Groundwork (Phase 1)

Composer file maintenance

- No locked packages (constraints)
- Well-maintained patch inventory
- Clean repository management

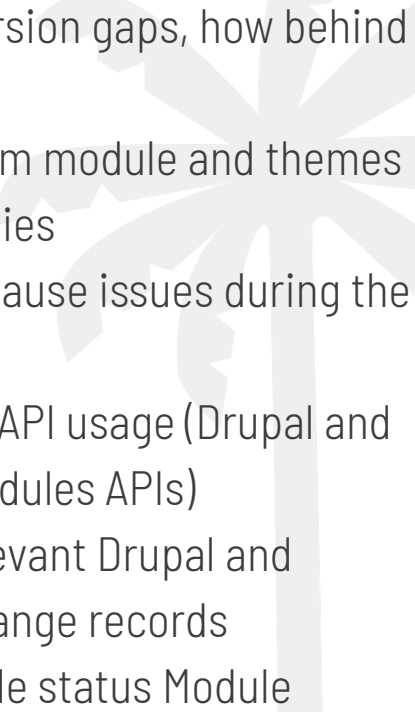
Think about configuration management strategy!



Laying the Groundwork (Phase 1)

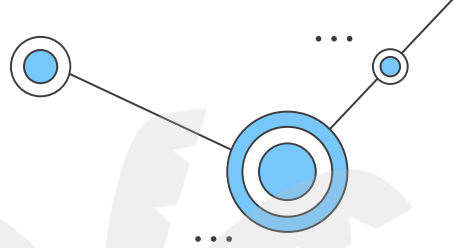


Initial assessment requirements

- Identify version gaps, how behind we are
 - Note custom module and themes dependencies
 - What can cause issues during the update
 - Document API usage (Drupal and contrib modules APIs)
 - Review relevant Drupal and module change records
 - The upgrade status Module
- 



The Update Cycle (Phase 2)

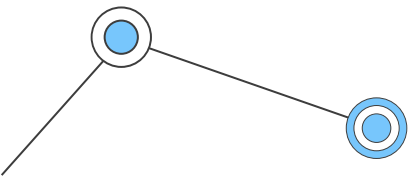


Standard update process:

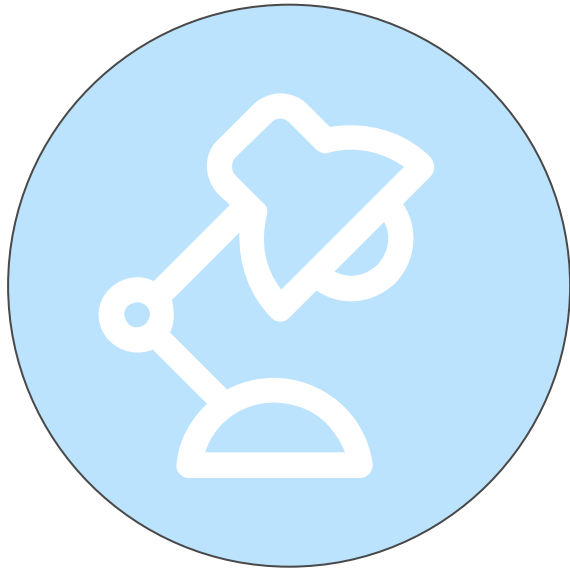
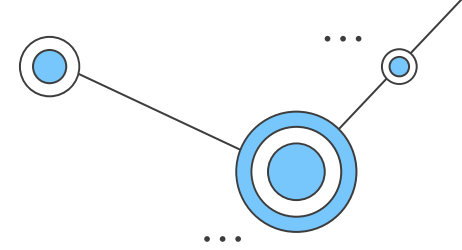
- Code updates via composer
- Database updates (drush updb)
- Configuration management
- Testing and QA verification

Key requirements:

- Match production code/database versions
- Follow systematic order (versions)
- Monitor update messages (for possible configuration changes)
- Export configuration when needed



Updates: A standalone Process



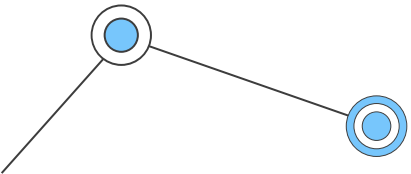
Not part of regular development sprints

Belongs in hotfix cycle

At stable points:

- Requires full release cycle
- Must be deployed to production
- Team synchronization required (local development):
 - New database copies
 - Updated code baseline

Prevents accidental overwrites of the update





Updates: A Time Travel Metaphor

- Each version exists in a specific point in time
- Module compatibility follows these timelines
- Multiple versions live simultaneously [Video Clip: 01:10:00]
- Shows practical example of version release date matching



3

From Process to Practice

How to do the updates

Understanding Update States

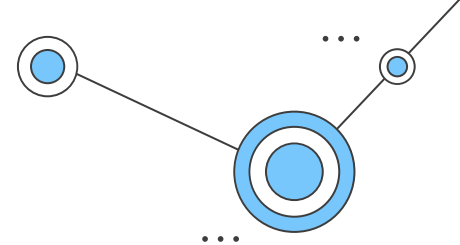
Stopping Points

- When issues need resolution
- Custom code compatibility
- Module version conflicts

Stable Points

- Clean update completion
- All tests passing
- Can be released [Video Clip: 1:43]

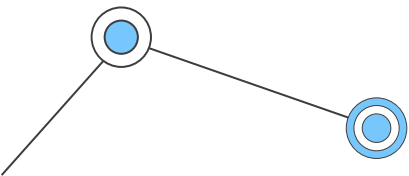
Managing Update Progress



At each stable point:

- Create database snapshots
- Commit code changes
- Document current state

Allows targeted rollback

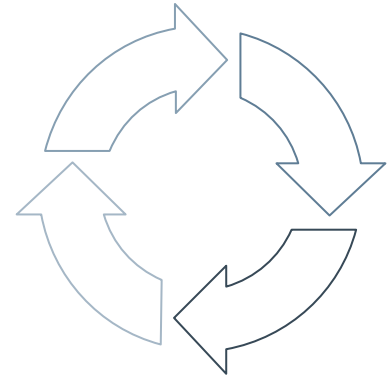


Deploying Updates

Updates are standalone processes

At stable points:

- Full release cycle required
- Complete production deployment
- Team synchronization, developers should:
 - Get a new database from prod
 - Rebase the updates code changes



4

A practical example

Time to do some updates!

Setting Up for Success

Environment configuration [Video Clip: 2:50]
Composer file cleanup:

- Moving from Drupal project to recommended project
- Handling installer paths
- Managing dependencies correctly

Critical environment requirements:

- Using DDEV or similar tool
- Proper PHP version management
- Running composer inside dev environment

Handling Update Challenges

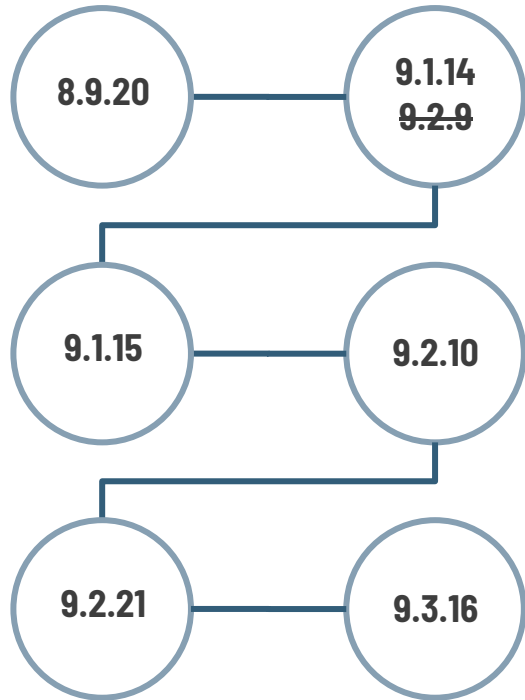
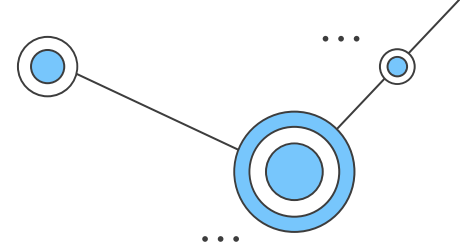
Systematic approach to errors:

- Reading composer conflict messages
- Identifying the real issue among multiple errors
- Solving one problem at a time

Real-world example:

- Identifying version mismatch
- Finding correct version based on dates
- Implementing the fix - Video Clip 27:45

Choosing the Right Versions



Version Release Date Matching technique:

- Finding core version release dates
- Matching module versions to same timeframe
- Understanding version compatibility windows

Workflow:

- Check core release date
- Find compatible module versions
- Verify module compatibility claims

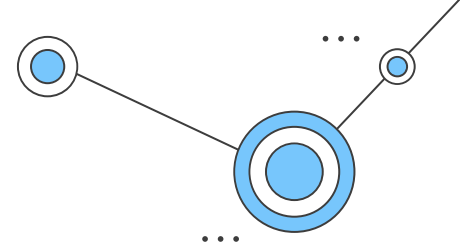
Video Clip 30:45





Best Practices & Takeaways

Keys to Update Success



Systematic preparation:

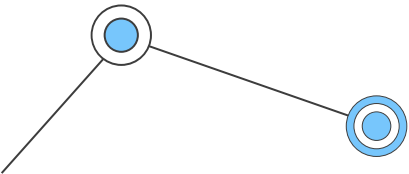
- Document all custom code and APIs used
- Maintain clean composer files
- Keep patch inventory updated

Strategic planning:

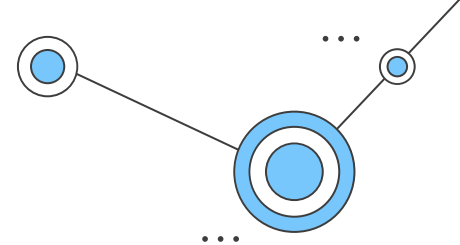
- Choose appropriate update windows
- Plan for adequate testing time
- Consider business impact

Testing priorities (Critical Path):

- Custom module functionality
- Theme compatibility
- Content workflows
- Integration points



Learning from Experience



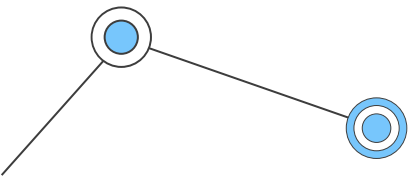
Version Release Date Matching technique:

- Finding core version release dates
- Matching module versions to same timeframe
- Understanding version compatibility windows

Workflow:

- Check core release date
- Find compatible module versions
- Verify module compatibility claims

Video Clip 30:45





Moving Forward



Tools demonstrated:

- Upgrade Status module
- Version Release Date Matching
- DDEV for environment management

Key takeaways:

- Updates as standalone process
- Importance of systematic approach
- Value of checkpoints and testing

Community resources:

- Drupal.org change records
- Issue queues
- Core announcements

¿Preguntas?

...

...

...

