# PEP003 Advanced Dependencies Update

# PEP003 – Advanced Dependencies

- Full dependency graph, instead of single list of dependencies
- Reuse existing installations
- Any compatible bundle may satisfy a dependency
- Pass a dependency output to another dependency
- Only installs bundles. Lifecycle management is a future PEP

getporter.org/proposals/src/pep/003-dependency-namespaces-and-labels.md

# Current Status

- Planning phase only as v1 of Porter and the Operator is our #1 priority
- We have an approved design for the user experience
  - Authoring a bundle with dependencies
  - Installing a bundle with dependencies
- Initial POC work is underway and incomplete
- Implementation is complex and is critical for our roadmap
- PEP needs implementation questions and decisions documented

# The Road So Far…

- Schema changes are already in v1 release
- Goal: incremental development, no big bangs, shipped in v1.x
- Refactoring in-progress
  - ☑ Query installations (prep to match existing installations)
  - ☑ Passed context and config to the porter.yaml to bundle.json converter
  - ☑ List tags from an OCI repository (prep to resolve latest available tag)
  - ⚒ Add feature flag
  - ⚒ Move existing dependencies into v1 package
- Playground branch where I'm vetting implementation decisions

# Playground Branch

- Define advanced dependencies in porter.yaml
- Represent as a new CNAB extension in bundle.json
- Generate a bundle dependencies graph
- Solve the graph
  - Reuse existing installations
  - Pick latest matching bundle from a registry
  - Check if a bundle satisfies a bundle interface
- 🚧 Generate an execution plan
  - Execution Order
  - Output passing
  - Identify what can run in parallel
  - JIT secret injection

# Workflows

- Execution plans are converted to a **Workflow**
  - Defines the tasks, inputs, and execution order
  - Decides what it will run in series or parallel
- Workflow is how that execution plan will run on a particular engine
- Stateful and persisted to Porter's database
- Users can see what has run, failed, is running, is pending
- Workflows can be restarted/retried when it fails

# Let’s talk workflow engines

- Executing dependencies (and bundles themselves!) are workflows
- Evolution of our internal workflow implementation
- Future work to secure mixins will use workflows too

# Can we use third-party workflow engines?

- Existing workflow engines
  - Kubernetes: Argo Workflow, Brigade
  - Docker: Cadence Workflow(?)
- More research needed
  - How to use JIT secret injection with third-party workflow engines
  - Do not copy and persist secrets out of a secret store
  - Understand security implications
  - Verify that our workflow plugin interface is possible with existing engines

# What's next?

- Refactoring and prep work continues
- Update PEP with implementation concepts and open decisions
- **Need your help vetting workflow engines and security**