The Phase2 Project Starter:

Octane

What is Octane?

Octane is a set of best practices and tools that Phase2 uses to kick start new projects quickly and efficiently. It represents years of collective experience acquired by hundreds of software architects, engineers, and developers at Phase2.

Octane consists of several loosely coupled packages:

- Common build scripts
- Support for git workflow
- Local development tools
- Remote environment management
- Continuous integration/deployment (CI/CD)
- Automated testing (functional, accessibility, visual regression)
- Support for Drupal and hosting providers

The primary benefits of Octane include:

Consistency: uniform project structure decreases the time needed to start new projects or onboard developers onto existing projects.

Speed and Efficiency: fast development workflow for building and testing increases sprint velocity and allows developers to stay focused on their current task.

Quality: standard tooling and process help ensure code quality and encourages re-use of best solutions.

Isolation: environments are created for each new feature or bug-fix allowing testing in isolation from other developers while still using common testing content.

Robustness: workflow and testing process keeps the primary environment stable and able to be released to production at any time.

Focus: allows developers to think less about "infrastructure" and focus more on the client-specific requirements of the project.

Flexibility: "loosely coupled" packages allow projects to extend and customize as needed to meet project needs.

Common build scripts

Common bash scripts are used in all environments (local, CI, etc) for various tasks:

Validate: ensures all code meets code style requirements regardless of language or framework.

Build: responsible for bringing in dependencies (composer, npm) and building any assets or artifacts.

Test: wraps various testing frameworks and groups them into a set of faster tests whose failure will "break the build", and a more comprehensive (and slower) set of tests that can be run on demand or scheduled as needed.

Git: provides scripts to support our git workflow.

DB: manipulates databases across environments.

CI: provides scripts to interact with the GitLab CI environments within our Phase2 DevCloud cluster.

Local Development

Octane provides configuration for spinning up local development environments via docker containers. Currently Octane utilizes the **Docksal** package:

- Provides a shallow wrapper around existing docker and docker-compose configuration and does not invent its own configuration language.
- Supports the concept of a "build container" (cli) that contains all of the tools needed to build a site (composer, npm, php, etc), allowing the web application container to only contain the needed apache or nginx images used in production.
- Supports persistent volume sharing between containers, along with managing the DNS of containers, allowing multiple projects to be running on a single computer in different namespaces.
- Command line interface (fin) implemented as a simple bash script without other dependencies.
- Supports the official Docker Desktop across both OSX and Windows (WSL2) platforms.

Because file sharing on the Mac OSX platform is very slow for both NFS and bind, Mutagen is used to synchronize files between the local file system and the docker containers. This achieves native file performance in both the local IDE and for the site and scripts running within Docker. By controlling which files are needed locally vs in the container, the lag between changing a local file and seeing that change in the container is minimal.

The Common Scripts mentioned in the previous section can be executed directly, or can be used via Docksal "fin" to validate, build, and test the site within the docker containers.

GitLab CI

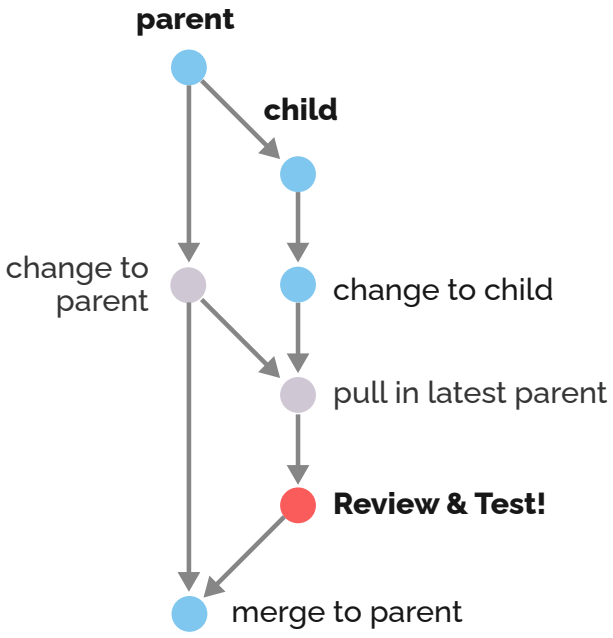
Octane currently uses GitLab CI for its deployment and integration testing. The same Common Scripts are used within the GitLab CI pipeline jobs to validate, build, and deploy the site to the Phase2 DevCloud.

Because the CI jobs use the same scripts used locally, developers who use the local validation and testing scripts rarely run into broken CI builds within GitLab. GitLab is also responsible for running the comprehensive testing scripts on a regular basis (usually nightly) to report any regressions that might not have been caught by the build tests.

Git Workflow

Octane supports our trunk-based git workflow:

- Each piece of work originates from a JIRA Issue, whether a new feature, bug fix, or task.
- A git branch is created for each issue.
- When commits are pushed, a remote environment for that issue branch is automatically created and deployed to our Phase2 DevCloud via GitLab CI.
- When a pull-request (merge-request in GitLab) is created for the issue, formal code review is done in GitLab.
- When the MR passes code review, QA testing is performed within the specific issue environment. This allows issues to be tested in isolation of other developers and issues.
- Once an issue has passed QA, the code is merged into the main branch and the issue environment is removed.



Git workflow (cont)

When the issue environment is first created, the environment (including database) is cloned from the main environment. This allows canonical test content to be created and maintained within the main environment and made available to each issue environment.

While GitLab is responsible for initially creating and deploying the environment to the DevCloud cluster, once an issue branch is created, subsequent commits to that branch trigger the validate, build, and test scripts running directly in the DevCloud environment.

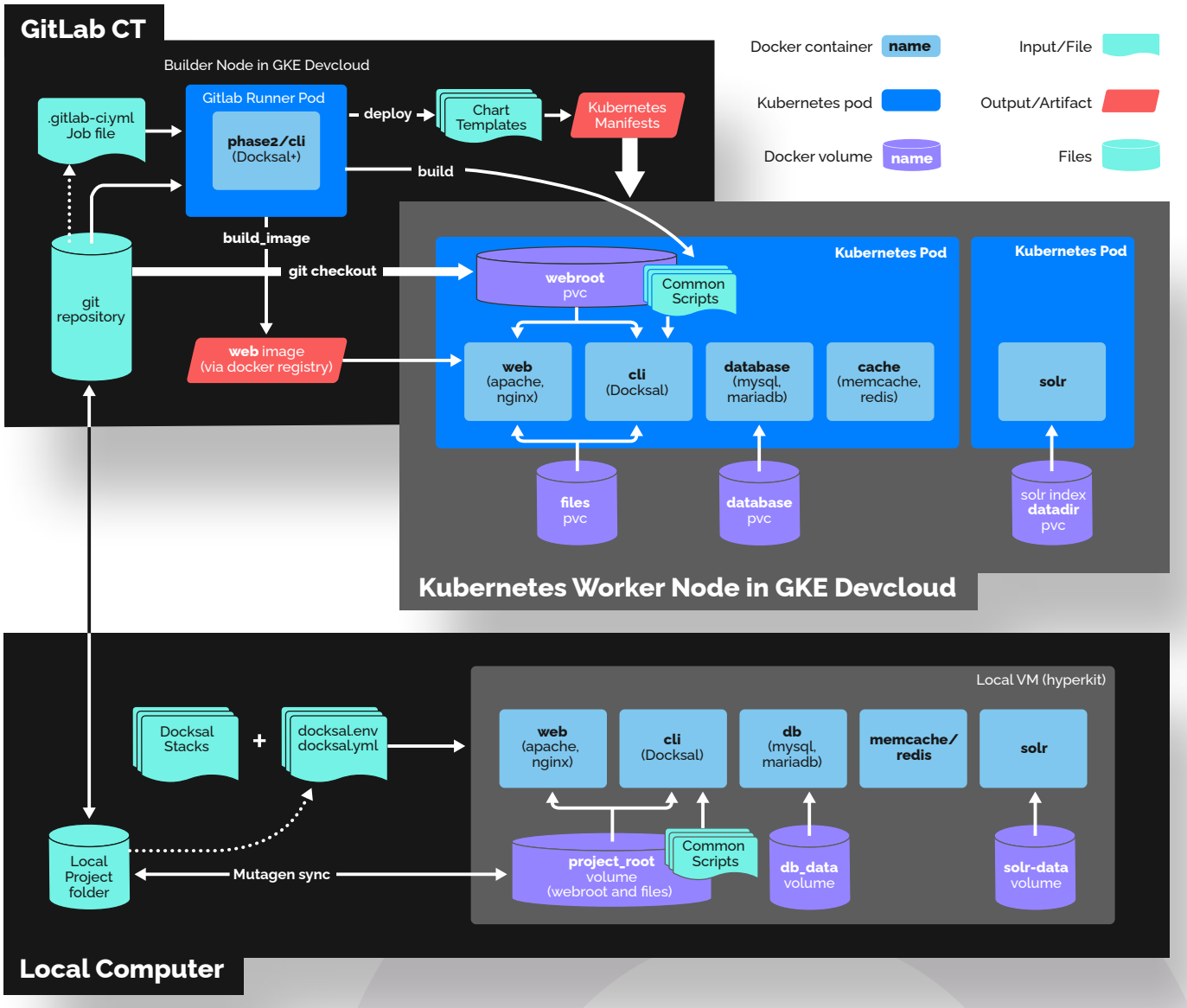
While the initial issue environment might take 2-3 minutes to be created and deployed, future commits are available within the remote issue environment in less than a minute. Validation and build failures are prioritized and can occur within seconds, allowing the developer to "stay in the zone" when working on an issue rather than waiting for a CI process and getting distracted by other tasks. Build tests try to stay under 5 minutes.

Phase2 DevCloud Cluster

Octane scripts create the Kubernetes manifest files (*.yml) that are used to deploy an environment to our Google (GKE) cluster. Typically, each project has a persistent main environment along with environments for each active branch being worked on.

Scripts are also provided to monitor the project environments within the cluster, connect to a shell container, examine log files, etc.

Octane Architecture



Testing Frameworks

Octane provides extensive help for testing. Scripts wrap several common testing frameworks for consistent use locally and in GitLab:

- **PHPUnit: unit, kernel, and Drupal ExistingSite tests allow a full range of Drupal PH code testing.**
- **Cypress: the primary functional testing framework. Plugins for AXE (Accessibility) and BackstopJS (visual regression) are also used.**

Fast tests, such as accessibility, unit, and kernel are performed within the DevCloud issue environments on every code commit/push to quickly identify problems and prevent failing code from being merged into the master branch. Slower visual regression testing is typically performed during the nightly comprehensive functional tests.

The ability to quickly and easily QA and run automated tests within each issue branch environment is one of the key aspects of the full Octane architecture. It focuses QA on the specific issue being addressed in isolation of other work being done, and keeps the main branch clean and ready for deployment to the client at any time.

Database Manipulation

The same common scripts used to move the database between the main environment and issue environments can also be used locally. Developers can easily pull down databases from any environment, or import databases to any environment, including their local. Backups of the main environment are made on every pull-request merge to allow for easy reverting if needed.

Integration with Outline

Phase2 uses its own Outline toolset for building and deploying packaged design systems of web components (html elements). Octane integrates with Outline by providing the tooling to bring the design system package into the Drupal site via npm. Common scripts are provided for updating and building the design system and Drupal theme both locally and in the GitLab CI environments.

Integration with Hosting

Octane provides deployment scripts for common hosting providers, including Acquia and Pantheon. Typically a project is still developed, integrated, and tested within the Octane GitLab CI environments. When a project is ready to push a release to the hosting provider, a release tag is created which triggers GitLab to build a clean deployment artifact that is pushed into the separate git repository of the host. Many projects trigger this nightly to update the host "dev" environment with the latest code on a regular basis. Because of our trunk-based git workflow and testing model, the main code branch is always clean and production releases can be done at any time and with any frequency needed by the client.

Drupal

While many aspects of Octane can be used for any application framework, the majority of projects using Octane are Drupal CMS projects. Octane includes a composer manifest for Drupal core and commonly used contributed modules. Starting configuration is also provided for common configurations such as:

- **Content editor experience (Layout Builder and associated modules and config, WYSIWYG)**
- **Search (Search API, Solr, etc.)**
- **Configuration management (config split, ignore)**
- **Media types (image, document, remote oEmbed, etc.)**
- **Theme support (Claro, Olivero)**

Octane is not a "distribution", but is a starter kit of building blocks for kickstarting a Drupal project.

Summary

Utilizing Octane, Phase2 has decreased the time needed to create a new client project in Drupal from days down to hours (or even minutes). Consistent tooling across our projects makes it easier for developers to move between projects or jump in and help with complex issues. The team can focus more time and resources on the specific client requirements of a project rather than with the DevOps tooling and infrastructure framework.

Phase2