
Introduction
This document explains how Platform.sh can positively impact your Drupal-centric project development process. Operational data from digital agencies and systems integrators deploying Platform.sh to deliver projects is showing savings of up to 25% in developer & systems administration effort. Delays currently experienced by development teams associated with the allocation of physical resources and provisioning of environments are largely eliminated.

Platform.sh is a general-purpose hosting platform for development, and can be used without limitation for a range of scenarios. From development through testing and into production, any permitted member of the team can replicate the entire technology stack end-to-end at the touch of a button. This eliminates many of the common causes of errors that often hamper code promotion through upstream environments, changing the way developers code and test even the smallest of features.

As a result, Continuous Integration (CI) processes are hugely improved and Continuous Delivery (CD) becomes an achievable reality for many organizations, enhancing their ability to respond to even the smallest of changes in the marketplace.

Environments are completely self-contained and can be easily branched, which makes testing, evaluation, and onboarding fast and safe.

![Diagram of environments](image)

*Figure 1: Environments are self-contained and can be easily branched*
Section 1. Key characteristics of the development environment

Platform.sh is a cloud based Platform as a Service (PaaS) designed for enterprise project development, with the following characteristics:

Remote development

Development environments are remotely hosted. This means that there is zero setup time when you on-board new developers, development environments are identical to each other and production, and all environments are fully backed up and unaffected by developer machine failure or loss.

On-demand development environments

Ad-hoc environments are created instantaneously and easily via the Platform.sh API. As such, they can be treated as disposable during development, testing, and deployment to production, radically improving the cost of change across most projects.

Discreet, isolated, complete

Environments are designed to be complete, self-contained and not share services. This has many advantages:

Environments can be identical, eliminating the difficult and familiar problem of code changes working on a development environment but failing on a test or production environment. Components can be upgraded without impacting other environments. For example, each environment can have its own search service with no danger of one results index contaminating another, or a heavy load on one environment bringing down the service for another.

Version upgrades on components are easier. For example, you can now easily have development, test, and production environments running different versions or configurations of PHP, MySQL etc. or different service configurations entirely, such as trying a new database, web server, or search technology.
Zero-touch configuration

A common development problem is one of configuration drift, where server configurations diverge over time. This can be due to any number of reasons such as bug fixes, minor or urgent releases, package updates and so on.

Most configuration tools in popular use (such as Puppet and Chef) only manage a subset of an environment’s state and so only partly solve this problem. Platform.sh deploys completely self-contained immutable environments to solve this, meaning that configurations literally cannot be changed manually, and are stored in configuration files and so captured, managed, version controlled, and deployed. The other benefit of this approach is a rapid commit-level rollback capability.

Section 2. Platform.sh - Technical Concepts

Components

Topology - A topology is a very simple XML-based configuration file which describes the make-up of an environment. It is managed alongside your code using Git.

Routes - Routes connect URLs to your environment and can be configured via the GUI or the API. For example, "mysite.com" might be served by PHP, but "mysite.com/push" is served by NodeJS, and "files.mysite.com" is mapped directly to a file system path.

Services - Services represent the individual components of your environment stack such as database, search, web server, and so on. Platform.sh currently supports PHP/Nginx, Solr, MySQL, and Redis. This list is growing as Platform.sh is deployed to more customers. Custom services can also be added as required.

Services SDK - Platform.sh comes with an SDK (software development kit), which allows you to easily create new services. These can be reused throughout your account and will inherit all of Platform.sh’s framework management services such as monitoring, backup, cloning, and restore. They can also be shared with others if you wish.

API - Platform.sh has been built from the outset as an API-centric technology, and you have access to the full API. This allows you complete flexibility and freedom in customizing and automating deployments.

Platform automatically rebuilds an environment from the configuration each time. This configuration is stored along with your code.

Platform comes complete with Git hosting, which has branch-based access so you can restrict groups of people to read or write access to specific branches.

Platform is hosted on AWS, designed with capability for customized hosting scenarios - such as connecting directly to your on-premise servers or a private cloud.

Platform plays nicely with other technology you might use, in-house or external. We give you access to the entire underlying API of Platform.sh. You can interact with Platform.sh via Git, GUI (Routes), CLI, or API. This allows you complete flexibility in deployment scenarios to fit your environments and specific needs.

For example, you can easily interact with Jenkins and hosted test services like Saucelabs or your ticketing system as part of a CI or CD process. Or perhaps you might want to interact with custom stub services on complex integrations.

Figure 3: Code versioning, automation and continuous integration. Platform.sh integrates with a range of CI and workflow tools for even greater efficiencies.

https://platform.sh
Underlying Technology

**LXC container technology** - Platform.sh uses LXC (LinuX Containers) to allow multiple environments to operate in isolation on shared hardware and to be deployed nearly instantaneously. Services run in containers, and each environment runs in an overlay network (within a VPC), allowing services full IP connectivity with each other within that environment.

**Amazon Web Services** - Platform.sh is hosted on Amazon Web Services (AWS), and custom deployments may be created to satisfy specific customer requirements, for example: location on a particular Region, deploy within your own VPC, connect directly to dedicated servers on your own premises, etc.

---

**PLATFORM FRAMEWORK**

**Figure 4:** Platform.sh leverages innovative, best-of-breed open source technologies to deliver next-generation PaaS services.
Section 3. Platform.sh Workflow

Workflows are created via Platform.sh's hierarchical environment structure. This allows workflows to be made using convention over configuration.

A simple branch command creates a new environment. Platform.sh is tightly integrated with Git and creates an environment per branch - so you can even setup new environments from within your IDE!

Developer permissions are restricted to the branch that they are in. This gives you flexibility around different roles in your organisation, for example, you might create a sandbox branch for new developers or customers.

Figure 5: In this example setup, we have environments for staging, penetration testing, dev master, development, feature branch, and migration. With each clone taking about 90 seconds, these could all be set up and ready for development in around 10 minutes!
Section 4. Platform.sh Workflow - Deployment

Code

Deployment is a case of simply pushing code via Git to the environment’s parent, and as with creating new environments, this can be done by a developer directly within their IDE. On each commit, Platform.sh completely rebuilds the target environment. This ensures that an environment matches the state of the code-base at any given moment, thus eliminating configuration drift, which is often cited as a significant cause of bugs upon deployment.

Database and Files

The database and file system are copied from the parent environment as required, but do not get pushed back up. This can be overridden in particular cases - such as migrations - although these are often run via in-flight migration scripts on production. It’s therefore important to recognize that the entire application environment has to be contained within code, which also ensures that best development practices are followed throughout the project lifecycle, eliminating another very common area of failure.

Conclusion: Revolutionizing PaaS, unleashing Agile

With its innovative, Git-powered automation, Platform.sh revolutionizes the PaaS landscape and positively impacts development processes, cutting up to 25% in developer & systems administration effort. By unblocking resources and easing provisioning, Platform.sh eliminates delays and allows Agile-based projects to flow more smoothly and efficiently.

For more information and a customized demonstration, Contact Us: Platform.sh/contact

https://platform.sh commerceguys.com