platform

Development Process Optimization

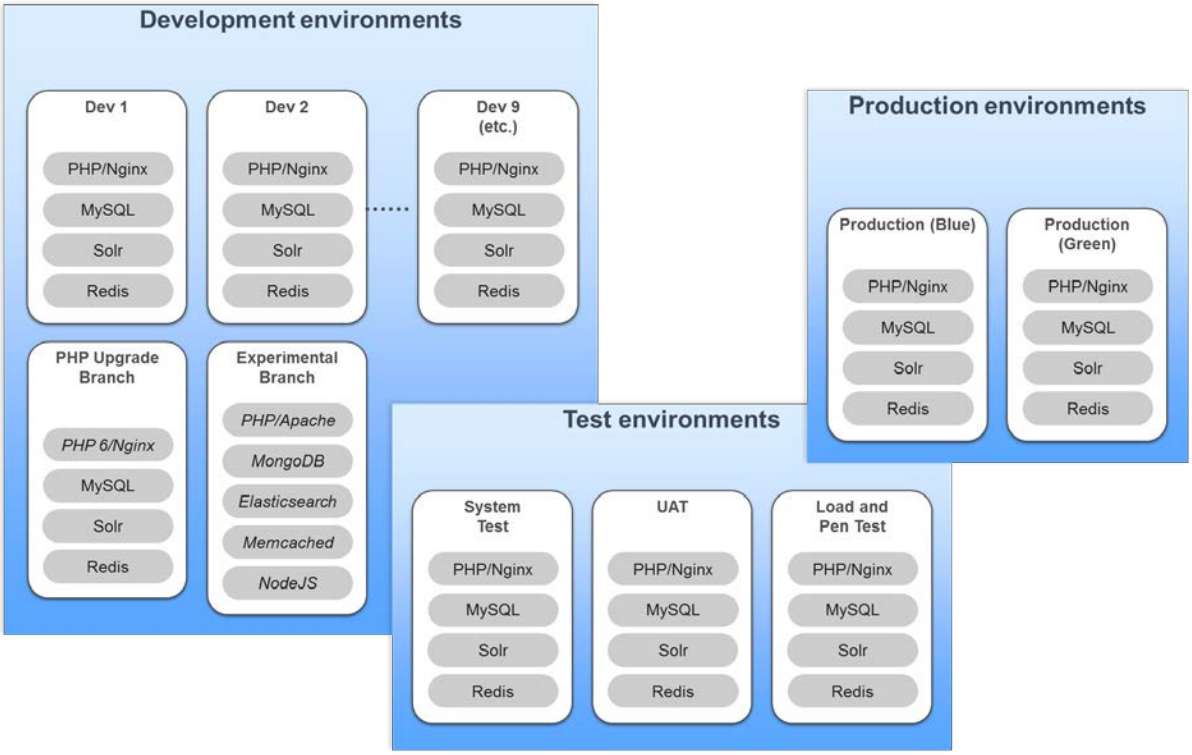
For Drupal centric projects

Introduction

This document explains how **Platform** impacts your Drupal centric project development process. Performance data from digital agencies and systems integrators utilising this offering to deliver projects is showing up to 25% in developer & systems administration resource savings. Delays currently experienced by development teams, associated with the allocation of physical resources and provisioning of environments are largely eliminated.

Platform is a general purpose hosting platform for development and can be used without limitation for many purposes. From development, through test into production, the entire technology stack is replicated end-to-end at the touch of a button, and by any permitted member of the team. This eliminates many common causes of error that we usually see when code is promoted through upstream environments, and changes the way developers code and test even the smallest of features.

Your Continuous Integration (CI) process is hugely improved and Continuous Delivery (CD) now becomes an achievable reality for many organizations, making their business functions highly responsive to even the smallest of changes in their marketplace.



commerce guys

Commerce Guys Inc.
950 Victors Way, Suite 10
Ann Arbor, MI 48108
t: 888 - 827-7142
www.commerceguys.com

1. Key characteristics of the development environment

Platform is a cloud based platform-as-a-service (PaaS) designed for enterprise project development, with the following characteristics:

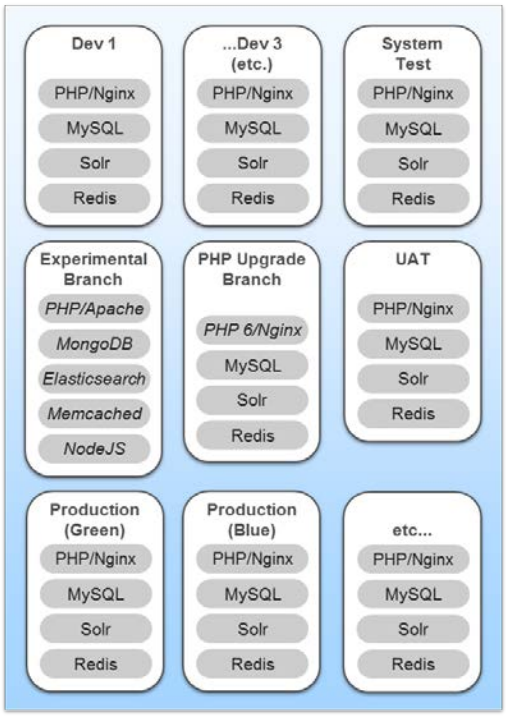
Remotely hosted development

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 our API. As such, they can be treated as disposable during development, testing, and deployment to production which radically improves the cost-of-change across most projects.

We have examples of development teams with previous access to only a handful of new environments during the course of a project, now creating and disposing of hundreds of environments each month as part of their typical development process. We designed the platform to be capable of supporting this.



Shared-nothing

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

- Environments can be identical - this eliminates the difficult and familiar problem of code changes working on a development environment but failing on a test or production environment.
- In addition, it allows you to upgrade components without impacting other environments. For example, each environment would have its own search service with no danger of one results index contaminating another, or a heavy load on one bringing down the service for another.
- This also makes it much easier to do version upgrades on components. 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, small/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. We use completely self-contained immutable environments to solve this, meaning that configurations literally cannot be changed manually, and are forced through configuration files and so captured, managed, version controlled, and deployed. The other benefit of this approach is a rapid commit-level rollback capability.

2. Platform – technical components

Components

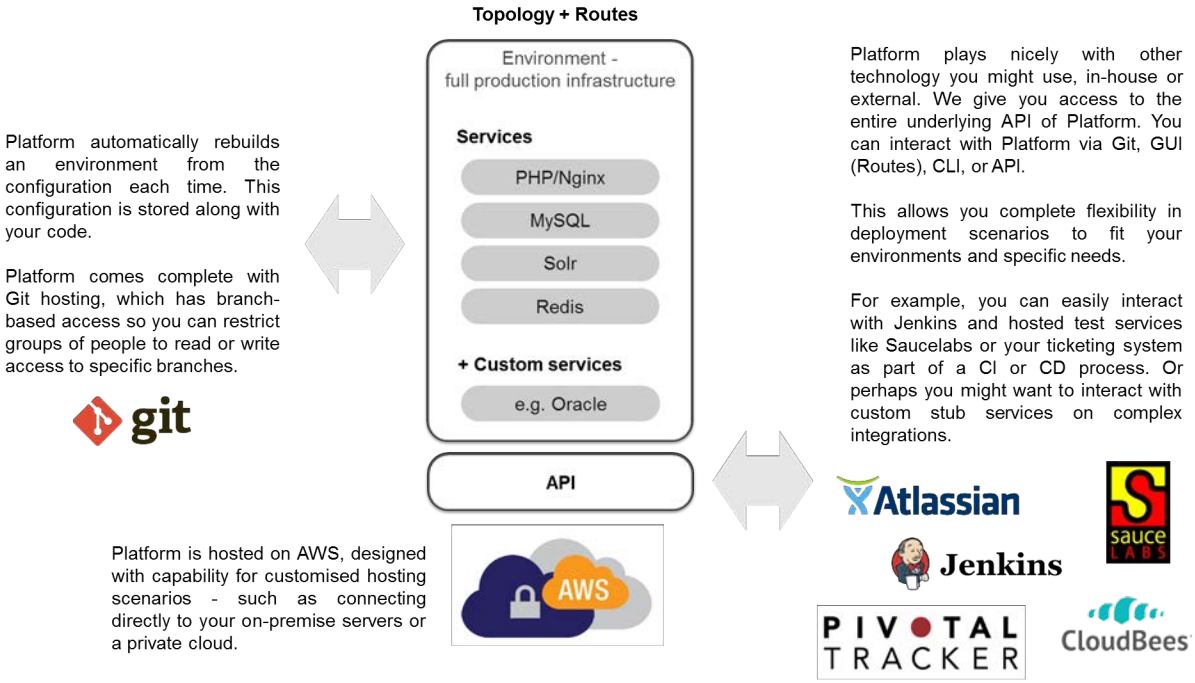
Topology - A topology is a very simple XML-based configuration file which describes the make-up of an environment. It is managed alongside your code using Git.

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. We currently provide services for PHP/Nginx, Solr, MySQL, and Redis. This list is growing as Platform is deployed to more customers and you can also create your own.

Services SDK - We provide an SDK which allows you to easily create new services. These can be reused throughout your account and shared if you wish. They will inherit all of Platform's framework management services such as monitoring, backup, cloning, restore etc.

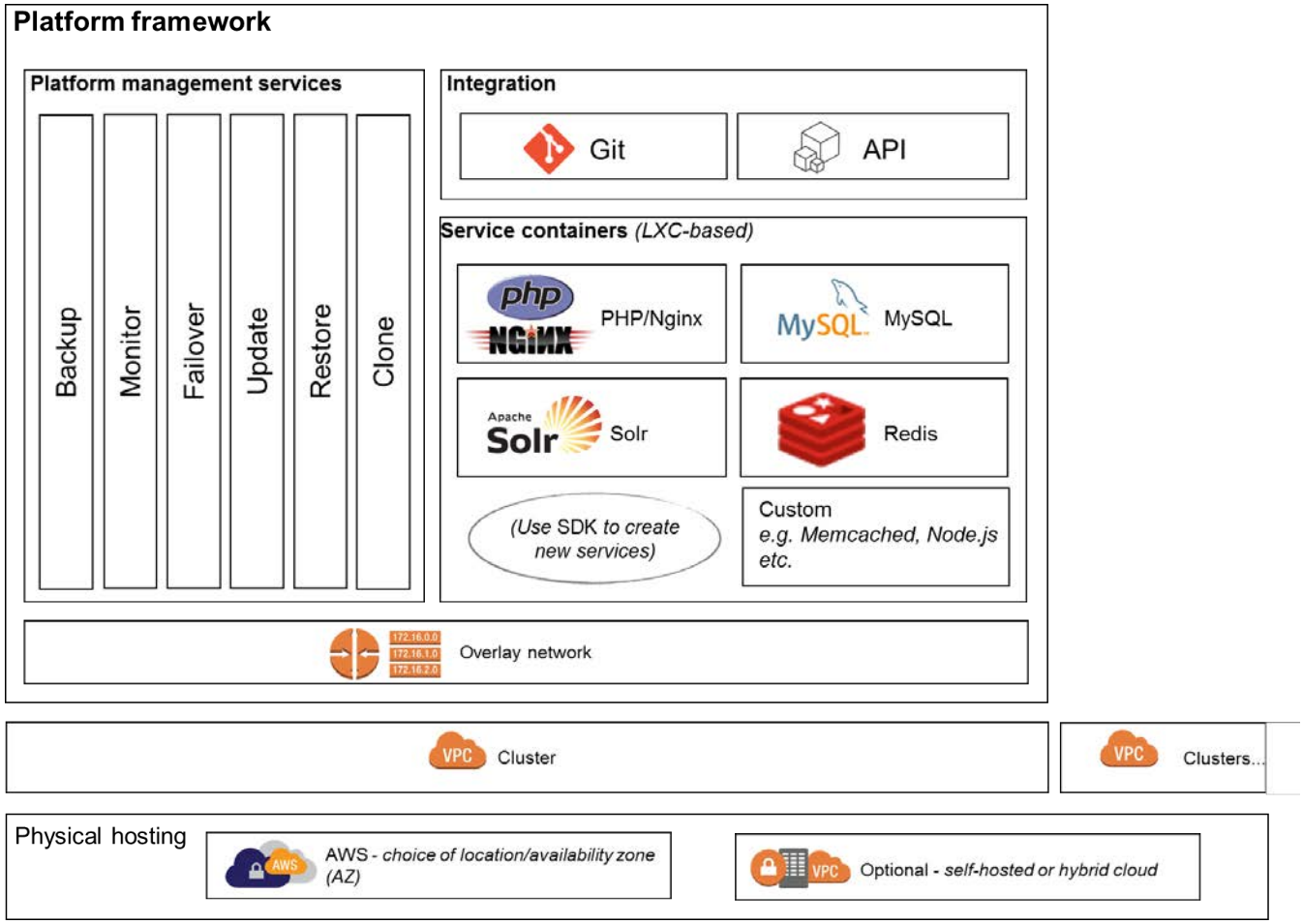
API - Platform has been built from the outset as an API-centric technology and you have access to the full API upon which we have built it. This allows you complete flexibility and freedom in customising and automating deployments.



Underlying technology

Platform uses LXC container technology to allow multiple environments to operate in isolation on shared hardware and to be deployed near-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.

Platform is hosted on AWS, and we can create custom deployments to satisfy specific customer requirements, for example if you need to be on a particular Availability Zone, or deploy within your own VPC, or you need to connect directly to dedicated servers on your own premises etc.



Commerce Guys Inc.
950 Victors Way, Suite 10
Ann Arbor, MI 48108
t: 888 - 827-7142
www.commerceguys.com

3. Platform workflow - Setup

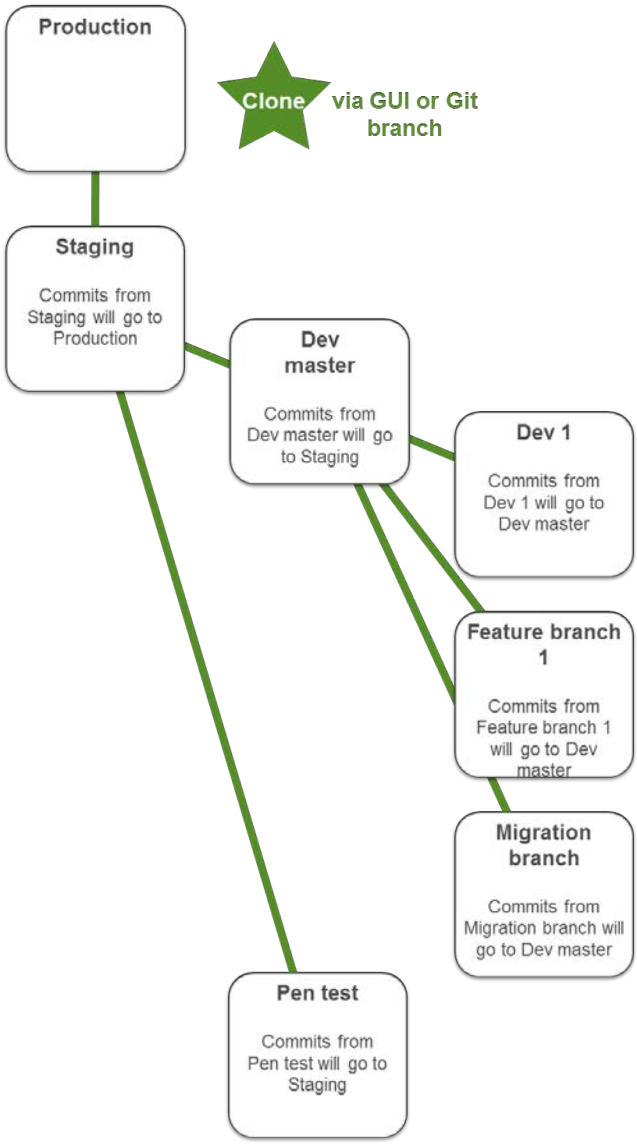
Workflows are created via Platform's hierarchical environment structure. This allows workflows to be made using convention over configuration.

A simple cloning command creates a new environment. Platform 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 clone branch that they are in. This gives you flexibility around different roles in your organization, for example, you might create a sandbox branch for new developers or customers.

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!



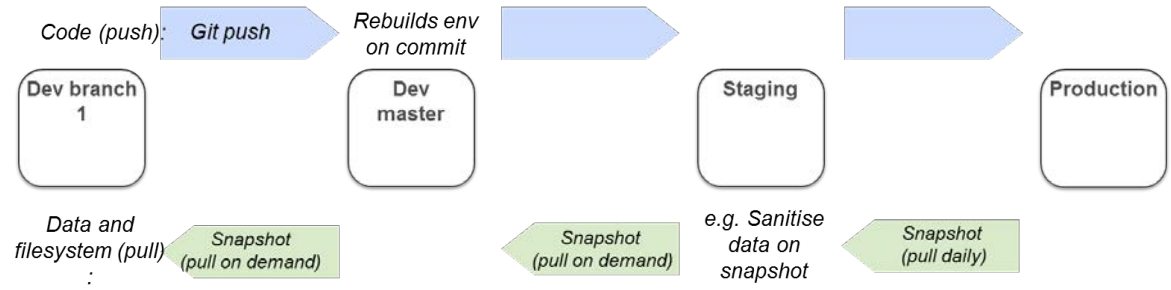
Commerce Guys Inc.
950 Victors Way, Suite 10
Ann Arbor, MI 48108
t: 888 - 827-7142
www.commerceguys.com

4. Platform workflow - Deployment

Code

Deployment is a case of simply pushing code via Git to the environment's parent, and as in creating new environments, this can be done by a developer directly within their IDE. On each commit, Platform 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 cited as the cause of the vast majority of bugs reported on deployment.

Example deployment workflow:



Database and files

The database and file-system are copied down 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 practice is followed throughout the project lifecycle, eliminating another very common area of failure.