WIKIPEDIA

Deployment environment

In **software deployment** an **environment** or **tier** is a computer system in which a computer program or software component is deployed and executed. In simple cases, such as developing and immediately executing a program on the same machine, there may be a single environment, but in industrial use the *development* environment (where changes are originally made) and *production* environment (what end users use) are separated; often with several stages in between. This structured release management process allows phased deployment (rollout), testing, and rollback in case of problems.

Environments may vary significantly in size: the development environment is typically an individual developer's workstation, while the production environment may be a network of many geographically distributed machines in data centers, or virtual machines in cloud computing. Code, data, and configuration may be deployed in parallel, and need not connect to the corresponding tier – for example, pre-production code might connect to a production database.

Contents

Architectures

Deployment architectures vary significantly, but, broadly, the tiers are bookended by starting at *development* (DEV) and ending at *production* (PROD). A common 4-tier architecture is *development, testing, model, production* (DEV, TEST, MODL, PROD), with software being deployed to each in order. Other common environments include Quality Control (QC), for acceptance testing; sandbox or experimental (EXP), for experiments that are not intended to proceed to production; and Disaster Recovery, to provide an immediate fallback in case of problems with production. Another common architecture is development, testing, acceptance and production (DTAP).

This language is particularly suited for server programs, where servers run in a remote data center; for code that runs on an end user's device, such as applications (apps) or clients, one can refer to the user environment (USER) or local environment (LOCAL) instead.

Exact definitions and boundaries between environments vary – test may be considered part of dev, Acceptance may be considered part of test, part of stage, or be separate, etc. The main tiers are progressed through in order, with new releases being deployed (*rolled out* or *pushed*) to each in turn.[1][2] Experimental and recovery tiers, if present, are outside this flow – experimental releases are terminal, while recovery is typically an old or duplicate version of production, deployed after production. In case of problems, one can *roll back* to the old release, most simply by pushing the old release as if it were a new release. The last step, deploying to production ("pushing to prod") is the most sensitive, as any problems result in immediate user impact. For this reason this is often handled differently, at least being monitored more carefully, and in some cases having phased rollout or only requiring flipping a switch, allowing rapid rollback. It is best to avoid a name like Quality Assurance (QA); QA doesn't mean software testing. Testing is important, but it is different from QA.

Sometimes deployment is done outside of this regular process, primarily to provide urgent or relatively minor changes, without requiring a full release. This may consist of a single patch, a large service pack, or a small hotfix.

Environments can be of very different sizes: development is typically an individual developer's workstation (though there may be thousands of developers), while production may be many geographically distributed machines; test and QC may be small or large, depending on the resources devoted to these, and staging can range from a single machine (similar to canary) to an exact duplicate of production.

Environments

The table below describes a finely-divided list of tiers.

Environment/Tier Name	Description
Local	Developer's desktop/workstation
Development/Trunk	Development server acting as a sandbox where unit testing may be performed by the developer
Integration	CI build target, or for developer testing of side effects
Testing/Test/QC/Internal Acceptance	The environment where interface testing is performed. A quality control team ensures that the new code will not have any impact on the existing functionality and tests major functionalities of the system after deploying the new code in the test environment.
Staging/Stage/Model/Pre-production/External-Client Acceptance/Demo	Mirror of production environment
Production/Live	Serves end-users/clients

Development

The development environment (dev) is the environment in which changes to software are developed, most simply an individual developer's workstation. This differs from the ultimate target environment in various ways – the target may not be a desktop computer (it may be a smartphone, embedded system, headless machine in a data center, etc.), and even if otherwise similar, the developer's environment will include development tools like a compiler, integrated development environment, different or additional versions of libraries and support software, etc., which are not present in a user's environment.

In the context of revision control, particularly with multiple developers, finer distinctions are drawn: a developer has a *working copy* of source code on their machine, and changes are submitted to the repository, being committed either to the trunk or a branch, depending on development methodology. The environment on an individual workstation, in which changes are worked on and tried out, may be referred to as the *local environment* or a sandbox. Building the repository's copy of the source code in a clean environment is a separate step, part of integration (integrating disparate changes), and this environment may be called the *integration environment* or the *development environment*; in continuous integration this is done frequently, as often as for every revision. The source code level concept of "committing a change to the repository", followed by building the trunk or branch, corresponds to pushing to release from local (individual developer's environment) to integration (clean build); a bad release at this step means a change broke the build, and rolling back the release corresponds to either rolling back all changes from that point onward, or undoing just the breaking change, if possible.

Testing

The purpose of the test environment is to allow human testers to exercise new and changed code via either automated checks or non-automated techniques. After the developer accepts the new code and configurations through unit testing in the development environment, the items are moved to one or more test environments.[3] Upon test failure, the test environment can remove the faulty code from the test platforms, contact the responsible developer, and provide detailed test and result logs. If all tests pass, the test environment or a continuous integration framework controlling the tests can automatically promote the code to the next deployment environment.

Different types of testing suggest different types of test environments, some or all of which may be virtualized[4] to allow rapid, parallel testing to take place. For example, automated user interface tests[5] may occur across several virtual operating systems and displays (real or virtual). Performance tests may require a normalized physical baseline hardware configuration, so that performance test results can be compared over time. Availability or durability testing may depend on failure simulators in virtual hardware and virtual networks.

Tests may be serial (one after the other) or parallel (some or all at once) depending on the sophistication of the test environment. A significant goal for agile and other high-productivity software development practices is reducing the time from software design or specification to delivery in production.[6] Highly automated and parallelized test environments are important contributors to rapid software development.

Staging

A stage, staging or pre-production environment is an environment for testing that exactly resembles a production environment.[7] It seeks to mirror an actual production environment as closely as possible and may connect to other production services and data, such as databases. For example, servers will be run on remote machines, rather than locally (as on a developer's workstation during dev, or on a single test machine during test), which tests the effects of networking on the system.

The primary use of a staging environment is to test all the installation/configuration/migration scripts and procedures before they're applied to a production environment. This ensures all major and minor upgrades to a production environment are completed reliably, without errors, and in a minimum of time.

Another important use of staging is performance testing, particularly load testing, as this is often sensitive to the environment.

Staging is also used by some organizations to preview new features to select customers or to validate integrations with live versions of external dependencies.

Production

The production environment is also known as *live*, particularly for servers, as it is the environment that users directly interact with.

Deploying to production is the most sensitive step; it may be done by deploying new code directly (overwriting old code, so only one copy is present at a time), or by deploying a configuration change. This can take various forms: deploying a parallel installation of a new version of code, and switching between them with a configuration change; deploying a new version of code with the old behavior and a feature flag, and switching to the new behavior with a configuration change that performs a flag flip; or by deploying separate servers (one running the old code, one the new) and redirecting traffic from old to new with a configuration change at the traffic routing level. These in turn may be done all at once or gradually, in phases.

Deploying a new release generally requires a restart, unless hot swapping is possible, and thus requires either an interruption in service (usual for user software, where applications are restarted), or redundancy – either restarting instances slowly behind a load balancer, or starting up new servers ahead of time and then simply redirecting traffic to the new servers.

When deploying a new release to production, rather than immediately deploying to all instances or users, it may be deployed to a single instance or fraction of users first, and then either deployed to all or gradually deployed in phases, in order to catch any last-minute problems. This is similar to staging, except actually done in production, and is referred to as a *canary release*, by analogy with coal mining. This adds complexity due to multiple releases being run simultaneously, and is thus usually over quickly, to avoid compatibility problems.

Frameworks integration

Development, Staging and Production are known and documented environments variables in ASP.NET Core. Depending on defined variable, different code is executed and content rendered, different security and debug settings are applied.[8]

See also

- Application lifecycle management
- Development, testing, acceptance and production
- Integrated development environment
- Software development

References

1. "Traditional Development/Integration/Staging/Production Practice for Software Development" (https://dltj.org/article/software-development-practice/). *Disruptive Library Technology Jester*. December 4,

2006.

2. "Development Sandboxes: An Agile 'Best Practice' " (http://www.agiledata.org/essays/sandboxes.html). *www.agiledata.org*.
3. Ellison, Richard (2016-06-20). "Software Testing Environments Best Practices" (http://www.softwaretestingmagazine.com/knowledge/software-testing-environments-best-practices/). *Software Testing Magazine*. Martinig & Associates. Retrieved 2016-12-02. "Once the developer performs the unit test cases, the code will be moved into QA to start testing. Often you will have a few environments for testing. For example you will have one set up for system testing and another that is used for performance testing and yet another that is used for user acceptance testing (UAT). This is caused by the unique needs for each type of testing."
4. Dubie, Denise (2008-01-17). "How to keep virtual test environments in check" (http://www.networkworld.com/article/2282388/data-center/how-to-keep-virtual-test-environments-in-check.html). *Network World, Inc*. IDG. Retrieved 2016-12-02. "Virtual server technology makes it easy for enterprise companies to set up and tear down test environments in which they can ensure applications will run up to par on production servers and client machines."
5. "Use UI Automation To Test Your Code" (https://msdn.microsoft.com/en-us/library/dd286726.aspx). *Microsoft.com*. Microsoft. Retrieved 2016-12-02. "Automated tests that drive your application through its user interface (UI) are known as coded UI tests (CUITs). These tests include functional testing of the UI controls. They let you verify that the whole application, including its user interface, is functioning correctly. Coded UI Tests are particularly useful where there is validation or other logic in the user interface, for example in a web page."
6. Heusser, Matthew (2015-07-07). "Are you over-testing your software?" (http://www.cio.com/article/2944759/agile-development/are-you-over-testing-your-software.html). *CIO.com*. IDG. Retrieved 2016-12-03. "Release candidate testing takes too long. For many agile teams, this is the single biggest challenge. Legacy applications start with a test window longer than the sprint."
7. Sharma, Anurag (2018). *Test Environment Management*. ITSM Press. p. 11. ISBN 9781912651269.
8. "Use multiple environments in ASP.NET Core" (https://docs.microsoft.com/en-us/aspnet/core/fundamentals/environments). *docs.microsoft.com*. Retrieved 2019-04-05.

Retrieved from "https://en.wikipedia.org/w/index.php?title=Deployment_environment&oldid=955042268"

This page was last edited on 5 May 2020, at 16:58 (UTC).

Text is available under the Creative Commons Attribution-ShareAlike License; additional terms may apply. By using this site, you agree to the Terms of Use and Privacy Policy. Wikipedia® is a registered trademark of the Wikimedia Foundation, Inc., a non-profit organization.