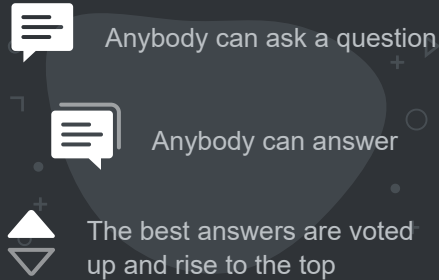


Software Quality Assurance & Testing Stack Exchange is a question and answer site for software quality control experts, automation engineers, and software testers. It only takes a minute to sign up.

Sign up to join this community



Software Quality Assurance & Testing

Why need two cycles of testing first on "QA Server" and then on "Stage server"?

Asked 3 years, 11 months ago Active 3 years, 11 months ago Viewed 6k times



5

We first get the code to be tested on the "QA server" and then on the "Staging server". Now when we are already done testing on the "QA server" why we can't simply push code directly to the "Production server".



I know its not safe and one more round of testing is always good, it's our gut feeling but if we go logically, why?



5

manual-testing



edited Jun 21 '16 at 7:39



Booakeater

856 1 7 15

asked Jun 20 '16 at 6:54



paul

609 9 18

1 What is the difference between QA server and Staging server in your company? Are they same environments? – [dzieciou](#) Jun 20 '16 at 9:09

It's not 'just one more round' it's different testing. – [Michael Durrant](#) Jun 21 '16 at 10:47

By using our site, you acknowledge that you have read and understand our [Cookie Policy](#), [Privacy Policy](#), and our [Terms of Service](#).

(more or less) Quoted:

5

QA stands for Quality Assurance. Probably the QA server is suitable for testing, measuring the quality of the software/hardware. Probably unit tests/regression tests are meant to run on this server.

The what-you-call staging/pre-production server, is a system running the production code used for regular usage of the software/hardware.

This means; you run the unit-tests against the source code on the QA server. This is a server that has software installed to run those tests, and more often than not it's a server that has a load of software running that might help with situations like mocking (thus you don't test it like you would in production, you fake a lot of stuff).

The Staging server, however, does not run tests against source code, but against production code. You compile or transpile the code. That means you can run integration tests and use an environment similar to the one used for production. That server should be identical to the production server in such a way that, when you deliver your product, you can be sure the production server is capable of handling the application.

edited May 23 '17 at 12:41



Community ♦

1

answered Jun 20 '16 at 9:12



Randy

174 ● 4

2 The "identical" part is so important here. If I had a dollar for every time a small data set on the QA server ran fine but sending it to prod and having the large quantities of data bog down a process, well I couldn't retire but I could take us to a mediocre lunch, assuming I skimmed a little on the tip. Still a few dozen times for sure... – [corsiKa](#) ♦ Jun 20 '16 at 17:04

2 -1 I would argue whether QA server is used for unit testing. Usually, unit tests are quite independent and isolated from hardware and software they run on and they are not run against any QA server on a QA server, they are run on build server. I would clarify precisely what you mean by unit testing here. – [dzieciou](#) Jun 21 '16 at 10:47

@dzieciou This may be simply my opinion. I read up on those when I tried to set up a CI environment for my project, and came upon that definition. Also, this [wikipedia](#) supports my claim. I know that's far from waterproof, but I sure didn't invent the definition myself. If you have a better way of explaining this, please create an answer. – [Randy](#) Jun 21 '16 at 10:56

@randy I wouldn't call Wikipedia always for credibility, especially when a section is labelled with "This section requires expansion." What I am saying is that it depends on the organization how they define QA environment and testing on that environment. I have provided my answer. And how does it work in your place? – [dzieciou](#) Jun 21 '16 at 13:26



4

Where I work, the environments are set up this way:

1. **Local systems** - this is where the devs build and do their initial unit testing. A number of service-based calls won't work in the local environment, and they are typically sharing the dev server's database with the dev environment, but they have enough to work with and build.
2. **Development server** - this is where the devs do integration testing and tests that work with the associated services for our products. It's more stable than the dev local systems, but still can be spectacularly broken. The database for this server is structurally correct but the data is sketchy and there are a lot of test and sample tables that have never been cleaned out.
3. **QA/TEST server** - this is where testers start: when changes reach this environment, they have been unit tested and integration tested by the devs. I test for correct function first, then correct handling of error conditions, and finally for edge cases. The most thorough and intense testing happens on this server.
4. **Staging server** - The staging server is set up to be as production-like as possible, including a recent copy of the production database with data cleaning to remove personally identifiable data and ensure the notification emails are not accidentally sent to live users. Unless code is being staged for deployment, the staging server runs the exact same code as the production server. When code is staged, I test happy path and any critical other paths I'm aware of for each work item (since at this point I am not checking functionality, I'm making sure that everything required for the change has been merged to the staging environment). I also run a small set of regression tests when I've done testing the merged code. *The main purpose of staging testing in this situation is pre-verifying the deployment process to make sure that all changesets are included and that nobody missed listing a changeset during development.*
5. **Production server** - Deploying to production is usually smooth and does not cause problems, because the staging process has caught all the glitches. I generally do a sanity check of the items being deployed and a quick run of a short regression suite immediately after deployment.

This process has evolved over several years to allow our deployments (a classic ASP web application) to run smoothly and cause customers minimal stress. Since I'm the only tester in the team and have been the driver for a number of the process changes, a lot of the effort is on me.

The only thing I'd change at this point is to automate the regression = but that's a long way off.

answered Jun 21 '16 at 16:00



Kate Paulk ♦

27.8k ● 6 ● 45 ● 96

Could you elaborate on what you would Chance? I find it hard to know the difference between unit and regression testing. I figure; you test something once and if it works, that test became a regression test. –

Randy Jun 21 '16 at 21:05

+1 This is **the** answer to this question – rsp Apr 9 '18 at 18:15

Excellent answer by Randy, but I'd like to go further : there can be **several** levels of testing machines, and each one goes further from the development, and looks like more the production machine. I've seen up to 4 levels, with

By using our site, you acknowledge that you have read and understand our [Cookie Policy](#), [Privacy Policy](#), and our [Terms of Service](#).





1. unit tests(someone, usually the developer itself, tests the isolated components, usually directly on the development machine)
2. integration tests (someone, maybe the developer, or a dedicated tester, tests the components together, usually on a dedicated integration machine)
3. performance and scenario-based testing(in France, we often call it "homologation"), where a dedicated tester tries to test the load-resistance of the application, while another tester(it's a different skillset, and usually different people) will execute realistic, complex scenarii.
4. acceptance testing, where a end-user is looking for things all people before might have missed, and that only a long-trained eye can notice.

Each step is designed to minimize the load of the following step. As you probably know, the earlier a mistake is detected, the less it costs to repair. the ideal is when end-users only detect mistakes noone else could have detected. If the end-user presses a button and the application crashes badly, someone in the steps 1, 2 or 3 made a mistake. When the only thing he finds is a complex, border-case that was not written in the specs nor reasonably guessable, then the steps 1, 2 and 3 have been done properly.

And I'm pretty sure some structures have even more layers. But you get the idea : each layer must detect as much defects as possible, but cannot detect everything.....hence the next layer.

answered Jun 20 '16 at 12:13



[gazzz0x2z](#)

436 ● 2 ● 7



0



QA and Staging server must consist on different environments. I worked in a company who has QA server and Staging sever. They make sure functionality is working properly on both environments or not. QA server is built for testing purpose. But I think Staging server is very similar to Product server.

answered Jun 21 '16 at 4:32



[Muhammad Ali Khamis](#)

529 ● 1 ● 9 ● 19



"consist on different environments."? what do you mean by consist on? – [dzieciou](#) Jun 21 '16 at 10:49



0



The QA server is typically used for running units tests that test components in isolation and that do not require other components or services.

The Staging server is typically used to test the full system including integrations and other services that are need and used in the full production system that it mimics.



The code base is the same (other than different branches perhaps) on both systems.


answered Jun 21 '16 at 10:42



[Michael Durrant](#)

By using our site, you acknowledge that you have read and understand our [Cookie Policy](#), [Privacy Policy](#), and our [Terms of Service](#).



- 1 Well, I worked in companies where both QA and Staging environment were having components integrated, just Staging had them more where QA was mocking them, and some configurations in Staging were closer to Production environment. In other word, QA testing was less credible. – [dzieciou](#) Jun 21 '16 at 10:51 
- 1 Also in some organizations Staging environment is visible to customers, so you need to be careful what and how you test. – [dzieciou](#) Jun 21 '16 at 10:56



0

The reason to test same code on different environments is because each environment has different characteristics that can help or limit performing different test scenarios and find different kinds of bugs. The way the company defines test environments depends on its needs, testing process and resources they can spend on it. Given that, what I describe here might not match what you have in your place:



QA environment:

- *less realistic*: many external systems might be mocked for different reasons (some infrastructure might be expensive to use or it might not been delivered yet)
- *less stable*, because frequent deployments of snapshots versions of software

Stage environment (called often Staging, CERT, or pre-production):

- *more realistic* as it is similar to production infrastructure
- *more stable* for many reasons: it already passed tests in QA environment and bugs in your code and code of other teams you rely on have been fixed; if it is visible to customers more effort might be taken by Operations department to keep it up and running; often used after code freeze, so no frequent deploys, unless a bug fix must be applied.

What realistic means here is that some end-to-end tests cannot be executed on QA environment, because there is no infrastructure required by those tests. Also, in QA you may not be able to find some integration bugs (false negatives, due to missing infrastructure) or bugs you find will not occur on Staging (false positives, if QA is mocking external systems in a wrong way).

Being more realistic, however, is not always a good thing. Staging is more like production so simulating some scenarios in Staging might be hard (if not impossible). Think, for instance, of payment system failure which is easier to simulate with QA environment with proper mocks.

Stability offers also different kind of trade-offs. More stable environment means your tests are more credible because their output is not dependent on system instability. However, you usually are obliged not to break system (perform negative scenarios) that would impact other teams using same environment.

There are companies that have also other environments in testing pipeline, both before QA and after Stage. In some companies, each team might have its own environment with external systems mocked (e.g., two teams may work on Web app: one team working on UI may have backed mocked).

In all places I worked for unit tests were not executed against any real environment with deployed system under test. Unit tests and some integration tests were rather mocking most system dependencies themselves at runtime.





9,654 ● 4 ● 37 ● 79

