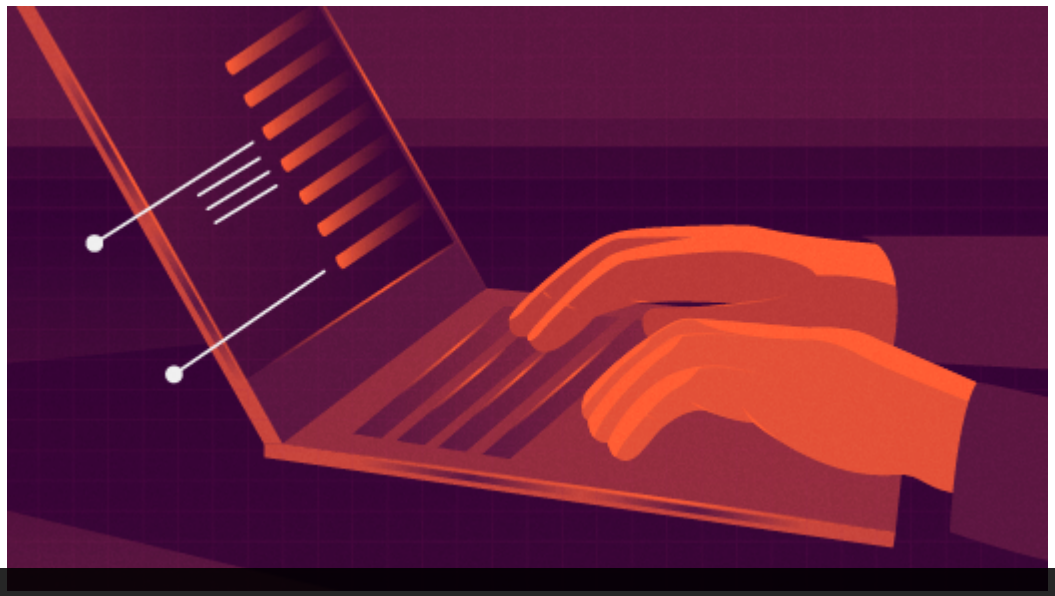
Don't test in production? Test in production!

Yes, testing in production is risky, but we should still do it, and not in rare or exceptional cases.

13 May 2019 | Ann Marie Fred (/users/annmarie99) | 135 | 4 comments



If you last updated your IT security standards five or more years ago, chances are they don't line up well with the realities of today's [DevOps and site reliability engineering (https://opensource.com/article/18/10/sre-startup)](https://opensource.com/article/18/10/sre-startup) (SRE) practices.

x testing in production—and, thus, testing with
/Ops and SRE blur the line between what is
at is a test and what is not.

ion, we'll dig into these questions:

st and production systems?
cording to the high standards of a production

ı systems so risky?
ıction?

- What about production data?
- How can we make testing in production less risky?

I should note that this is an opinion piece; it's based on years of collective DevOps and testing experience, but it's not to be read as an official IBM statement.

Why do we separate dev/test and production systems?

It's standard practice to treat development, test, and production systems differently, at least from a compliance and risk-management standpoint, mostly because they have differing security, data, and privacy controls. Let's take a step back and think about the historical reasons for the different attitudes toward these deployment environments.

- DevOps monitoring tools guide (https://opensource.com/downloads/devops-monitoring-guide?src=devops_resource_menu4)
- Getting started with DevSecOps

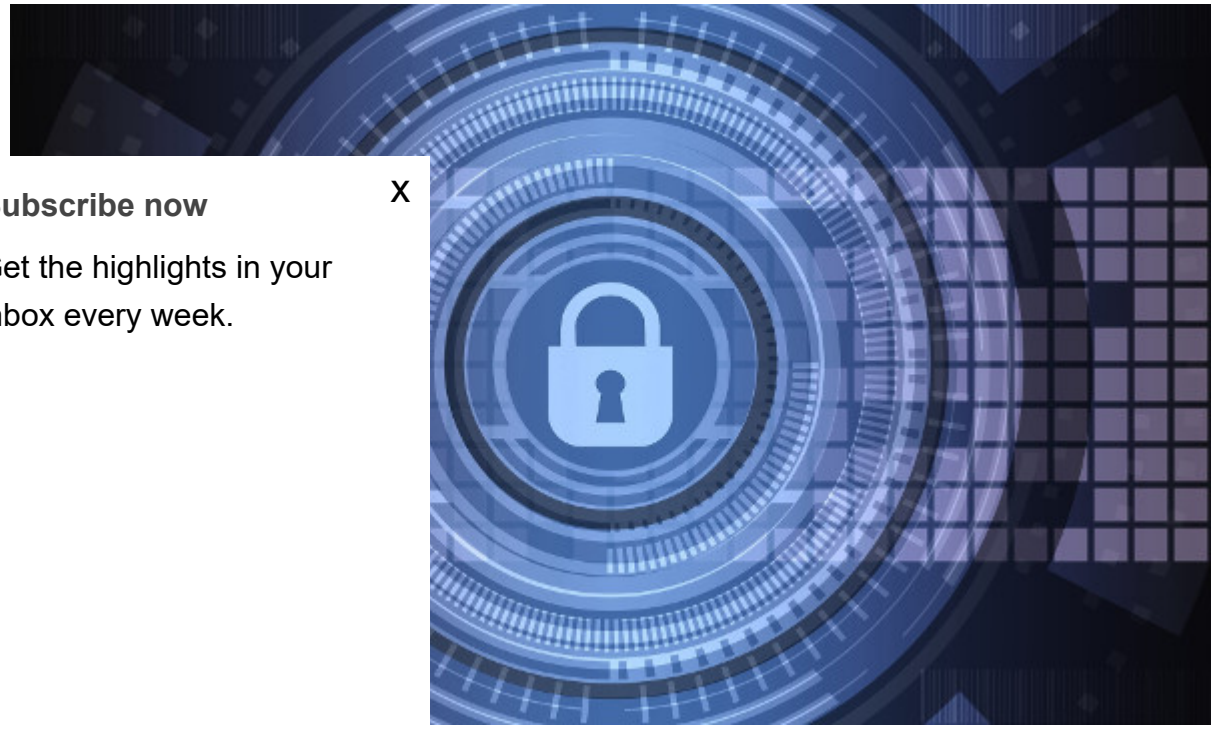
wnloads/devsecops?src=devops_resource_menu5)

concepts
om/article/2019/8/devops-terms-10-essential-
ırce_menu6)

s://opensource.com/tags/devops?
ı2)

e most important because they run our businesses
ems serve our customers and have a direct impact on
ıal for a developer's working environment to be "broken" for a few hours now and then, but we must manage production systems according to impeccable standards of quality, reliability, and availability. That's why it's crucial to limit risks to our production systems. DevOps and SRE still focus on risk avoidance, but they use different risk-reducing strategies, as compared to other practices (such as ITIL).

In addition, production systems are special because they have access to production data. Production data must be a reliable source of truth, so we must protect it from corruption. Production data is also likely to contain information that we can share only with authorized users, such as confidential or personal data, so we must ensure that it's protected by production-level authentication and authorization. Finally, we may need to maintain an audit trail of accesses to production data (create, read, update, and delete), which isn't needed for dev/test systems.



Production systems are more tightly controlled and monitored, for good reasons.

We also need excellent visibility into and control over the current state of our production systems. We monitor them carefully so we can quickly detect problems, and when problems occur, knowing the current configuration of those systems makes it easier to recover quickly. Most people couldn't care less whether a developer changes the configuration settings on their personal laptop, but we lock down production systems to a known configuration and with secure change controls in place. Whether we lock down the configuration via a change-control database or infrastructure-as-code, the goal is the same: visibility and control.

Finally, remember that we manage dev/test and production systems differently because there are compliance rules and regulations specifically for production systems. Few things kill velocity quicker than unnecessary burdens on our dev/test environments!

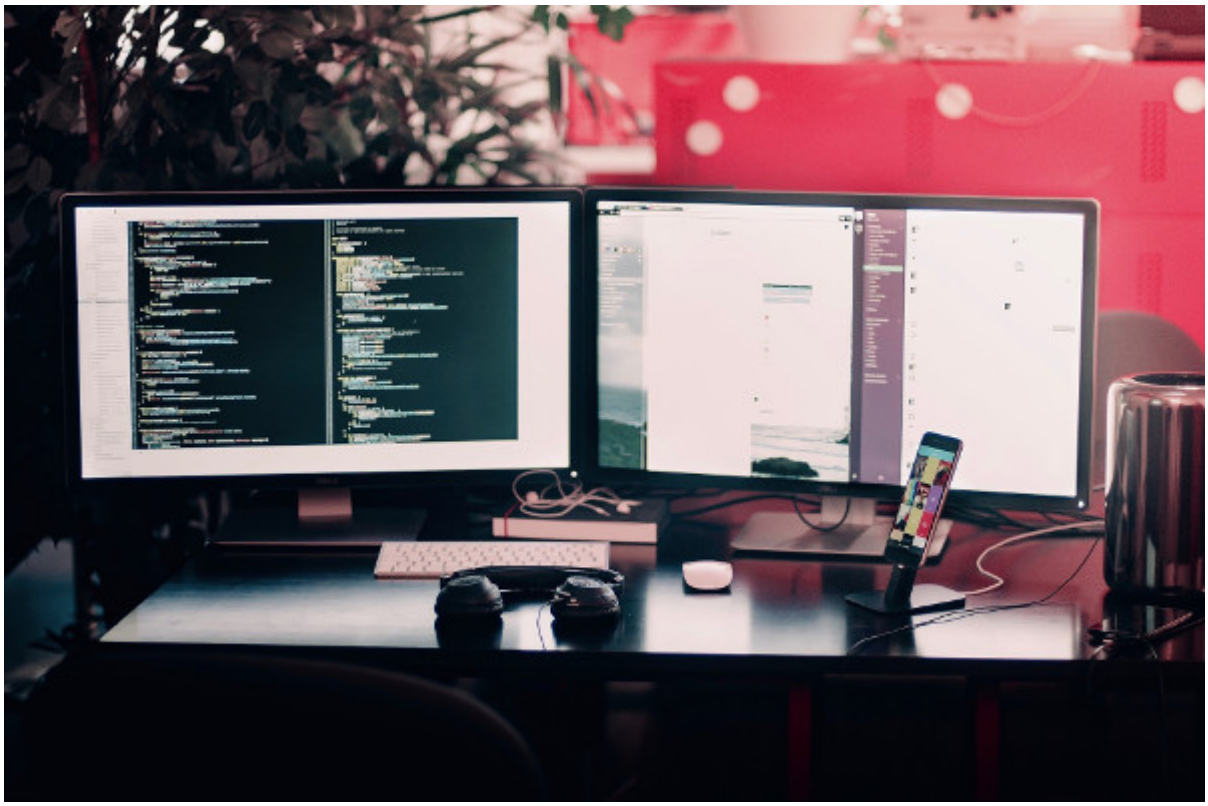
What should we manage according to the high standards of a production system?

When we started thinking about testing in production, we quickly realized that we were beginning with [illegible] should be obeying [illegible] and

what is not a production environment. But, as happens with most assumptions, we were wrong. Developers and testers want to move fast; when in doubt, we tend to classify systems as dev/test instead of production so that we don't have to deal with

ement overhead. But how do we know when we need ace? It's not all black-and-white, but there are several

amples: We can agree that developer laptops and ally designed for testing (e.g., integration test, system production systems. Furthermore, there's a general erve real customers using real data, directly or tion systems. There are also systems that we only l enough to the operations of the company that they systems.



Modern software development and delivery practices can blur the line between development, test, and production systems.

Often, though, the line between "production" and "non-production" depends on your unique situation and so your use of the terms

- Staging
- Pre-production
- Pre-live

example, might be one that you run only tests
e of a test system. On the other hand, your staging
ur business partners use to test new APIs before you
should manage it like a production system, for most
you expect it to simulate the real user experience of
erate a bit more downtime for that type of server, but
lity authentication and authorization; you should put
rations, and you should monitor the servers like a

A content management system's preview environment is another example of a system that sounds like one type but is really the other. Preview content is not published yet. Maybe it's time-sensitive too, such as a website for an unannounced product. Someone will publish the new product's web pages for all the world to see after the announcement; but before publication, they are highly confidential. Therefore, the preview environment must have even more authentication and authorization controls than the production environment. It must not render a preview page unless the current user has the right to see it.

We should treat these like production systems too:

- Blue/green deployments. Why? The backup environment that is not getting traffic could become the production environment at any moment.
- Backup servers in a high-availability configuration. Why? The backup servers could start serving production traffic at any time.
- Canary deployments. Why? These serve a small portion of production traffic.
- Staged rollouts. Why? All versions of the hardware and software that are serving production traffic are "in production."
- A/B testing. Why? Every "B" is in the mix with production traffic too.

It's important to be consistent when you apply rules and heuristics to your systems and environments. You shouldn't consider a staging environment a production system one day and a test system the next. That's a recipe for disaster. Make sure

f the systems are production systems and which are
s decisions and any exceptions.

g which systems are production systems and which
ropriately, will ensure that you protect your
rting your development and test velocity.

oduction systems so risky?

n production," it's because they want to avoid several

- Corrupted or invalid data
- Leaked protected data
- Incorrect revenue recognition (canceled orders, etc.)
- Overloaded systems
- Unintended side effects or impacts on other production systems
- High error rates that set off alerts and page people on call
- Skewed analytics (traffic funnels, A/B test results, etc.)
- Inaccurate traffic logs full of script and bot activity
- Non-compliance with standards

Why should we go ahead and test in production anyway?

Yes, testing in production is risky, but we should still do it, and not in rare or exceptional cases, either. These tests-in-production are accepted as best practices in the DevOps and SRE communities:

- Usability testing and UX research

- Final smoke testing of blue/green deployments
- Feature flags
- Staged roll-outs

- ...duction system monitoring, including scripted health
- ...web pages to compare staged vs. production
- ...ing (after initial testing and deployment)
- ... for broken links and report errors
- Other testing of high-availability/disaster recovery plans
- Bug bounty programs

Production tests help us:

- Prevent bad deployments from breaking production systems
- Objectively identify which user experiences are more effective
- Design more delightful user/site interactions
- Gradually roll out new features
- Get quick feedback on success or failure of our latest changes
- Catch problems before users notice them
- Understand web page performance characteristics and change impact
- Build more resilient systems
- Improve system quality

By running several types of production tests, either at deployment time or on a frequent schedule, we can cover a variety of critical non-functional requirements:

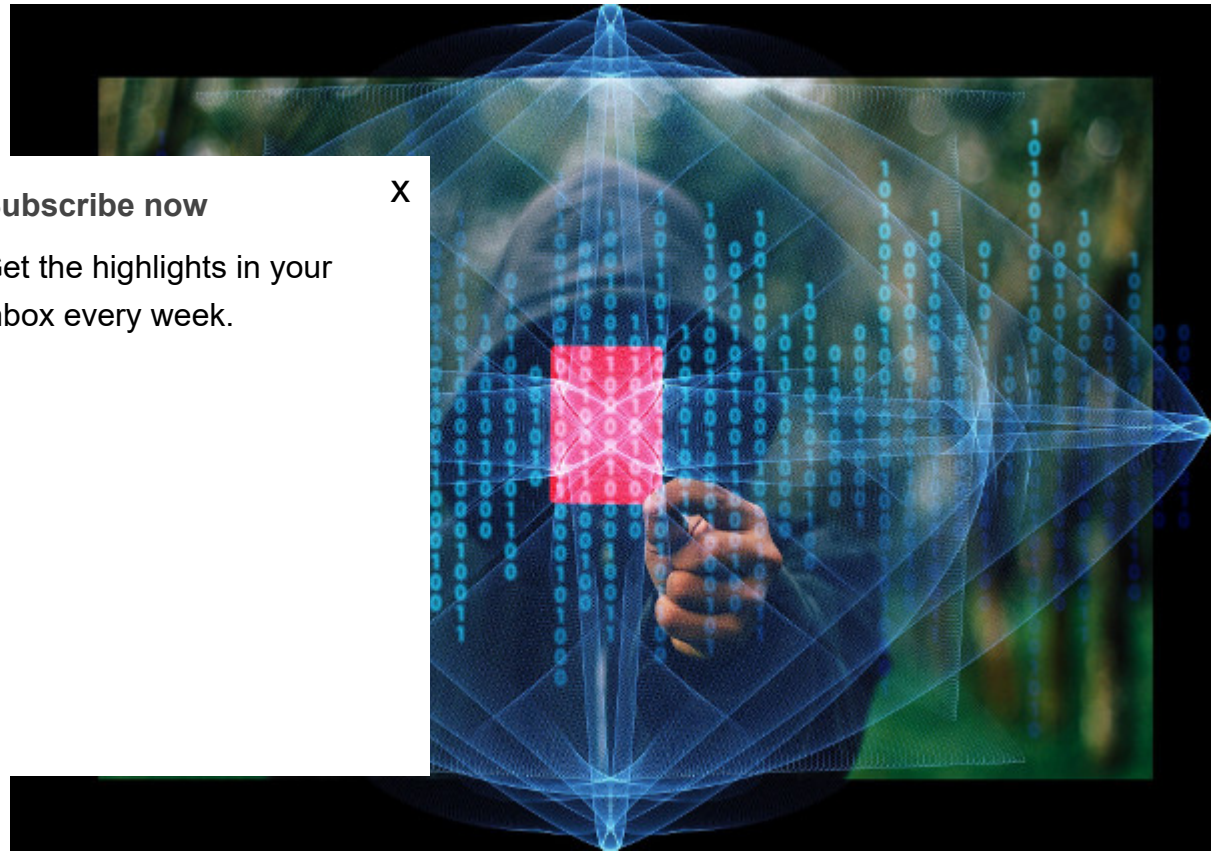
	User experiences	Availability	Roll-out changes	Feedback	Quality	Performa
				✓		
				✓	✓	
		✓	✓		✓	
			✓	✓		
			✓	✓		
testing		✓	✓	✓		
Health checks	✓	✓			✓	
Regression testing	✓				✓	
Broken-link checkers	✓				✓	
Real-user monitoring	✓			✓		✓
Chaos engineering		✓			✓	✓
Failover testing		✓			✓	
HA/DR testing		✓			✓	

Penetration testing		✓			✓	

s of various types of production tests

't there always a problem child?): Third-party ould we do it on production systems? On the one example, if pen testers find an injection vulnerability, ed data in your database. On the other hand, hackers ng suites on your internet-facing systems every s happening on many of your production systems ot. That's why I've included it in this list of production tions:

- Make sure that your pen testers-for-hire are working with a production-like environment and not a toy.
- Run the most popular security test suites against your test systems and fix any errors you find before everyone else runs the same tests against your production systems.



Your production systems need to be resistant to hacking attempts and handle them gracefully.

Finally, did you notice that these tests-in-production have something in common? None of them makes "test copies" of production data. They all operate directly on *real* production systems and data.

What about production data?

Here's a shortcut: Dev/test environments may not need special test data. They can often use production data that's available to anyone, like actual web page content, as long as your tests won't modify that data, saving you the time and expense of creating test data. All of the production tests in the previous section fall into this category, as do many web services and APIs.

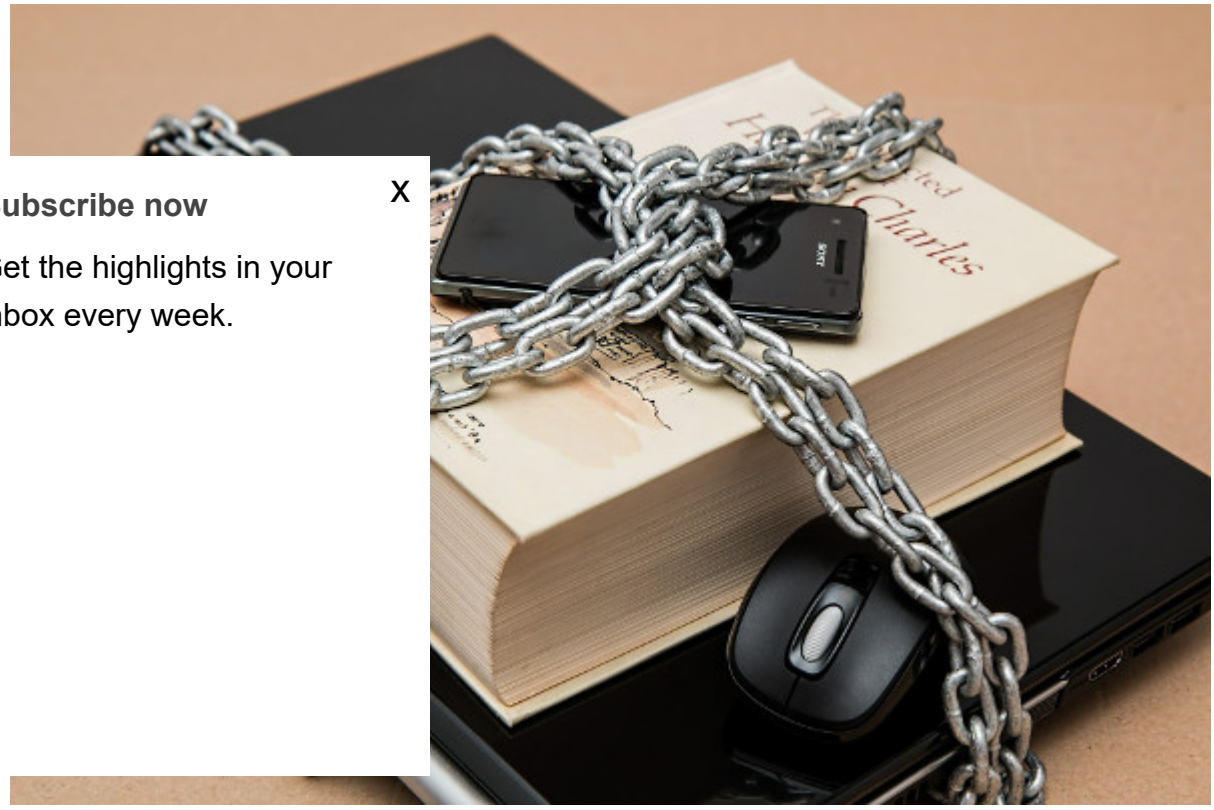
But beware! Just because data is available on the internet or a REST API is free to use doesn't mean you can use it for your dev/test purposes. Make sure you understand and comply with any applicable license agreements and website usage

It's great if you can save time and money by using production data, but some of your applications and services need to modify your data store, so you'll need tests that modify data as well. Running these tests in production is difficult to do without

tems. Faced with this reality, when it becomes tore in order to validate a test scenario, most have different data sources: one for dev/test and you set up test data that's realistic and complete

small enough, you could technically make a copy of oduction data into a dev/test environment is ass security and privacy controls. (GDPR eneral_Data_Protection_Regulation), anyone?)

Let's take an example. You've put into place your carefully thought-out security and privacy controls. You set up your production systems so that only people with a "need to know" can access any personal data; you know where your data is stored; you've established a process for removing personal data from your systems on demand; and so on. Maybe you have customer addresses and phone numbers in your database. If someone copies the database to a dev/test system, and you didn't implement your security and privacy controls there, you have a hole in your system. If a customer exercises their "right to erasure" and requests that you delete their address and phone number, how will you know which dev/test systems you need to update to remove that information? What if a developer's laptop or a test mobile device with personal data on it is stolen? Will you have to report and mitigate a security breach? To close these holes, you need to either keep personal data off of your dev/test systems or include dev/test systems with access to personal data in your production data compliance scope.



Locking your laptop and phone with a chain isn't the best plan. Assume these devices will be stolen eventually and decide how to manage them accordingly.

The obvious alternative is to use mock data for dev/test environments, but deciding when to use mock data for testing is difficult because creating and maintaining mock data is time-consuming and error-prone. If you start with a production database and manually scrub it to obscure or remove sensitive data, you might miss something. If you go too far with a scrub, you might corrupt the data or limit its testing usefulness. Conversely, if you build up a test database from scratch, it's difficult to create all of the permutations and edge cases you need to include for a good test suite.

Only you and your teammates can decide which direction is the right one to take for your unique purposes, but here are some considerations that will help:

- Understand what controls are on your production data before you use it and respect those controls.
- Make sure that your entire team agrees on the meaning of *sensitive data* and *personal data*.

- Define and get acceptance of your protocols for handling sensitive and personal data.
- Document and understand the sensitive and personal data in your systems.

- bers and testers understand exactly what sensitive

duction data to remove personal or sensitive process for sanitizing the data isn't itself a hole. Consider third-party scrubbing and sanitizing kely to cause errors than a homegrown solution.

sting in production less risky?

test in production" is to protect our production systems. Now that we've established that we can and should test in production every day, how do we keep our production systems safe?

Have a testing plan and complete it before putting a new component into production. Well-tested code is less likely to fail in production.

First and foremost, test all systems thoroughly with automated tests before you get to production. I'm a firm believer in 100% automated unit test coverage, with unit [illegible] you should complete multiple layers of testing before going to production.

functional/behavioral testing, integration testing, deployment/configuration testing, accessibility testing, security testing, and high-availability failover testing. And if you're doing gradual roll-out of new features, test the roll-out process too. And yes,

but never as a replacement for test automation!

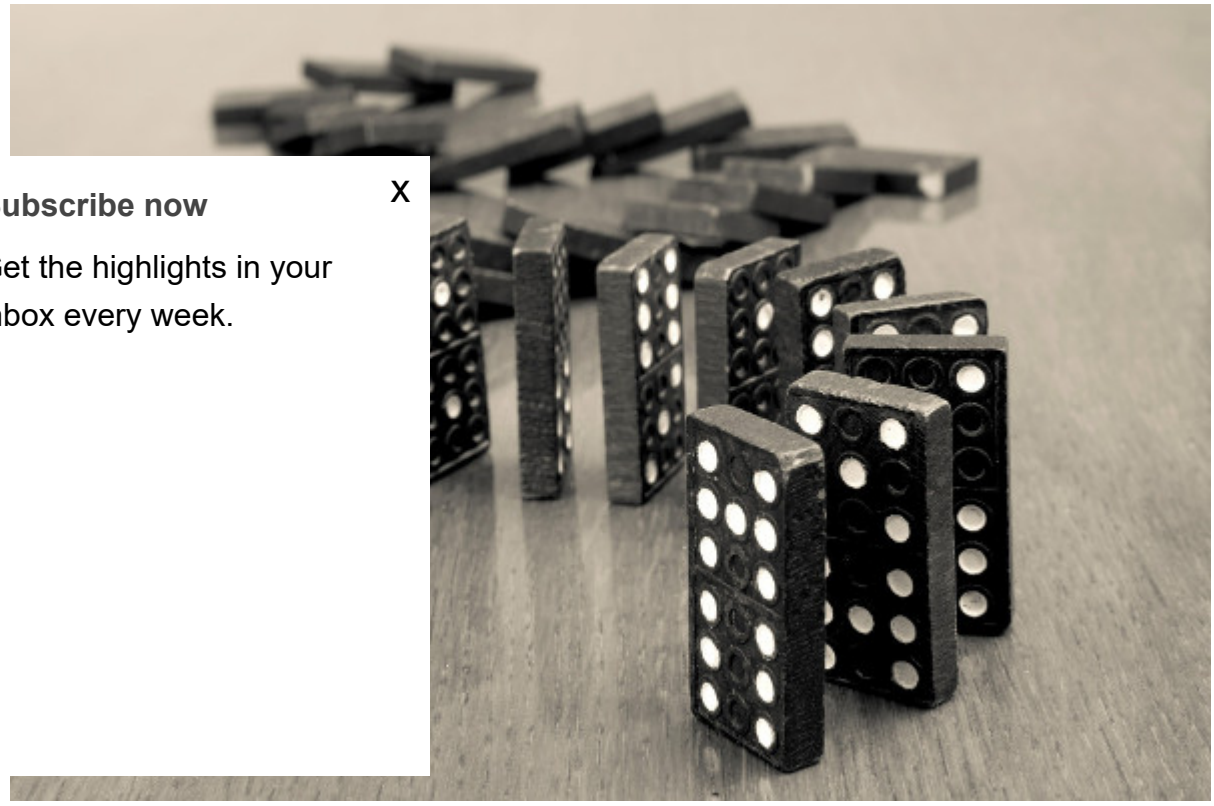
ı production? Two words: Be careful. "Bug hunt" ɔractice where you ask everyone on your team to trying to find bugs in your software. These are fun ɔu set up the proper guardrails. Review with your ɔduction and teach your bug hunters what they should sing their personal credit card and immediately ɔf tests can interfere with revenue recognition and

Also, don't reconfigure your production systems manually, not for testing purposes nor for any other reason. Manual changes leave your systems in an unknown state, and it can be very difficult to get them back to a known configuration. Manage your production systems' configuration using infrastructure-as-code (Chef, Helm, etc.) and/or release management and orchestration tooling, like IBM UrbanCode Deploy or Kubernetes (/article/17/6/introducing-kubernetes).

Before any chaos engineering, make sure you've planned your experiments in keeping with the basic principles of chaos engineering (https://en.wikipedia.org/wiki/Chaos_engineering):

1. Plan an experiment
2. Contain the blast radius
3. Scale or squash

You can also reduce the risk of chaos engineering by meeting these prerequisites: solid automated test coverage, good monitoring and alerting, a high-availability setup with fast automated failover, and a team that's on-call and ready to restore service within your service-level objectives (SLOs) if something fails. Within my team, we normally run the first fail-over tests for each component on a weekend with the developer on call present, and teams "graduate" to testing for resilience during normal working hours after they pass the first set of tests.



Don't let chaos engineering, scalability, or performance testing cause a cascade of problems in your system. Loosely coupled components, good error handling, and planned failure modes help to isolate failures.

When planning scalability and performance testing, make sure that you won't impact your customers. Don't throw a bunch of API requests at your production system and hope for the best. Use a separate, isolated environment if the cost-benefit analysis justifies it. If you need to test scalability or performance in production, ramp up traffic gradually while monitoring your systems, and stop before service disruption or failure. And don't forget to filter out scalability/performance test traffic from your production analytics!

These risk-reduction techniques will help you keep your production systems resilient and less likely to fail due to testing in production.

Conclusion

Testing in production is extremely valuable and a best practice in modern software

- Prevent bad deployments from breaking production systems
- Objectively identify which user experiences are more effective
- Design more delightful user/site interactions

es

ess or failure of our latest changes

s notice them

rmance characteristics and change impact

l testing in production; rather, we should understand

eguards into our systems to address them. We

y and compliance standards to take modern production testing practices into account.

Thank you to the attendees of the "test in production" open space at Devopsdays Charlotte (https://devopsdays.org/events/2019-charlotte/welcome/), who collaborated to brainstorm and distill what we really mean by the terms "test" and "production" and what we really need to do to protect our production systems. I would also like to thank Craig Cook and Jocelyn Sese for their helpful feedback on early drafts of this article.

Topics : DevOps (/tags/devops) CI/CD (/tags/cicd)

(/users/annmarie99)

About the author

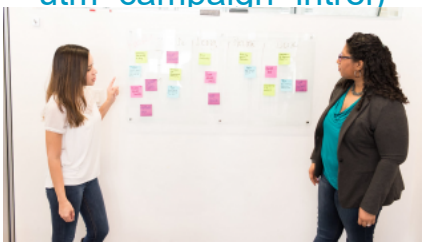
Ann Marie Fred - I'm a DevOps Lead, software engineer, ops enthusiast, former manager, client advocate, fixer, and champion of best practices. I've been a software engineer for 20 years and a DevOps practitioner for 8 years. I've worked in research, consulting, web portal development, IT systems management development, cloud computing, hybrid cloud, deployment automation, and most recently, web platform development and operations. My specialties are DevOps, continuous delivery, security, compliance, cloud...

More about me (/users/annmarie99)

11 DevOps lessons from My Little Pony (/article/20/5/devops-lessons?utm_campaign=intrel)

DevOps vs. Agile: Do they have anything in common? (/article/20/4/devops-vs-agile-common?utm_campaign=intrel)

(/article/20/5/cicd-best-practices?utm_campaign=intrel)



How does kanban relate to DevOps? (/article/20/4/kanban-devops?utm_campaign=intrel)

Getting started with Jenkins Configuration as Code (/article/20/4/jcasc-jenkins?utm_campaign=intrel)

How to be the right person for DevOps (/article/20/3/devops-relationships?utm_campaign=intrel)

4 Comments

Evilbastard (/users/evilbastard) on 15 May 2019 1

An excellent setup would have two production systems that mirror each other , you can then deploy changes from Development/Testing/QA to one mirror, and if everything checks out make that the default production, putting the other mirror into the "Testing" phase.

Ann Marie Fred (/users/annmarie99) on 16 May 2019 1

A blue-green deployment is a fully automated version of the production mirror setup.
https://docs.cloudfoundry.org/devguide/deploy-apps/blue-green.html

dfoundry.org/devguide/deploy-apps/blue-green.html)

of the production environment, but a blue-green deployment
deployment and testing is automated, so the tests either fail or
ites, and the cut-over to the new environment is fully automated and

rentaaronreed) on 21 May 2019 1

iece to scaling DevOps and we are using this as a piece at our
tup, thanks so much, Ann Marie! I shall endeavour to get some

Ann Marie Fred (/users/annmarie99) on 29 Jul 2019 0

Glad to hear it!