DevOps Testing Micro-Credential

Syllabus

Table of Contents

DevOps

5 Introduction
7 The DevOps Pipeline
10 DevOps Testing
13 The Role of Automation in DevOps Testing

References

16 ISO/IEC/IEEE Standards
16 Trademarks
16 Books
17 Other References

General Information

STUDY TIME – 200 MINS.

KEYWORDS

behavior-driven development, build, commit, continuous delivery, continuous deployment, continuous integration, continuous monitoring, continuous testing, DevOps, DevOps toolchain, infrastructure as code, test-driven development

LEARNING OBJECTIVES FOR DEVOPS

Introduction

(K1) Recall the purpose of DevOps
(K1) Recall the benefits of DevOps

The DevOps Pipeline

(K2) Compare the differences between Continuous Integration, Continuous Delivery and Continuous Deployment
(K2) Explain the concept of Continuous Testing
(K1) Recall the purpose of Continuous Monitoring
(K2) Describe the full DevOps pipeline

DevOps Testing

(K1) Recall the concept of testing during planning
(K2) Understand the difference between TDD and BDD
(K2) Understand how unit testing and integration testing are applied in DevOps
(K2) Describe the types of testing that occur during staging and deployment

The Role of Automation in DevOps Testing

(K2) Understand the relationship between continuous testing and automation
(K2) Describe infrastructure as code
(K2) Explain the DevOps toolchain

Introduction

DevOps bridges the gap between development and operations by bringing the two teams together for the entire software lifecycle, from development to delivery. DevOps is a cultural shift focused on building and operating at high velocity. It is an approach that involves activities that are "continuous": continuous development, continuous integration, continuous testing, continuous deployment, continuous delivery, and continuous monitoring.

Development roles in DevOps include everybody who is involved in making the software and everyone who is involved in running and maintaining production. DevOps asks people working in the development and operations teams to work together, from the beginning of a software development project until it is delivered, breaking down the walls that stand between the different departments.

The benefits include:

Assessing quality at every stage of development and operation, resulting in a high-quality product with fewer defects

Releasing small increments frequently – If a good DevOps pipeline is set up, it helps to release the product more often, in small increments, which provides early and frequent feedback

On-time (and possibly lower cost of) delivery, in part due to a better delivery process

Reduction in vendor and third-party issues

Faster time to market

There are some misconceptions about DevOps. For example, DevOps does not eliminate any roles from development or operations. DevOps is not a separate team – it is a culture shift. DevOps is not a tool, nor is it the use of fancy tools without a defined process.

The DevOps Pipeline

A DevOps pipeline is set up in every project to aid all the continuous activities.

Continuous Integration (CI): Continuous integration is the practice of merging all developers' working code into a main branch of the codebase in a shared repository several times a day. The developers' changes (commits) are validated by creating a build and running automated tests against the build. Continuous integration ensures that the application does not break whenever new commits are integrated into the main branch of the codebase. If problems are introduced with the new code, they are quickly identified and resolved.

Continuous Delivery: Continuous delivery builds on the CI process in which teams produce software in short cycles, ensuring that software can be reliably released at any time. In addition to having automated builds and tests, Continuous Delivery requires an automated release process and a way to easily deploy applications at any time.

Continuous Deployment (CD): Continuous Deployment (CD) takes the continuous delivery process one step further. A change that passes all stages of the DevOps pipeline is released to the customer. The deployment trigger is automatic; therefore, the whole release process is automated.

Note that continuous integration (CI) is part of continuous delivery and continuous deployment. Continuous deployment is continuous delivery when the releases happen automatically.

Continuous Testing: Continuous testing is the repeated execution of tests against a codebase. It is the quality gate throughout the DevOps pipeline and increases confidence in the product. The success of continuous delivery or deployment relies on continuous testing. At every stage of the DevOps pipeline, continuous testing ensures that

what is being delivered through a stage of the DevOps pipeline is correct and good enough to progress to the next stage of the pipeline.

For continuous testing to succeed, the siloed testing and operations teams should be integrated with the development team. Testing should not be a separate activity. Instead, it should be incorporated as much as possible into the DevOps pipeline. For example, performance and cybersecurity testing should not be an independent activity but should be incorporated into the pipeline. The DevOps pipeline should not be bypassed or blocked for a long time. Use of drivers and stubs are highly encouraged for testing any "what-if" scenarios and dependencies.

Continuous Monitoring: Continuous monitoring allows the DevOps team to constantly monitor the application in a production environment to ensure that the application is performing at an optimal level and the environment is stable. Continuous monitoring helps in diagnosing and fixing errors as soon as they are found.

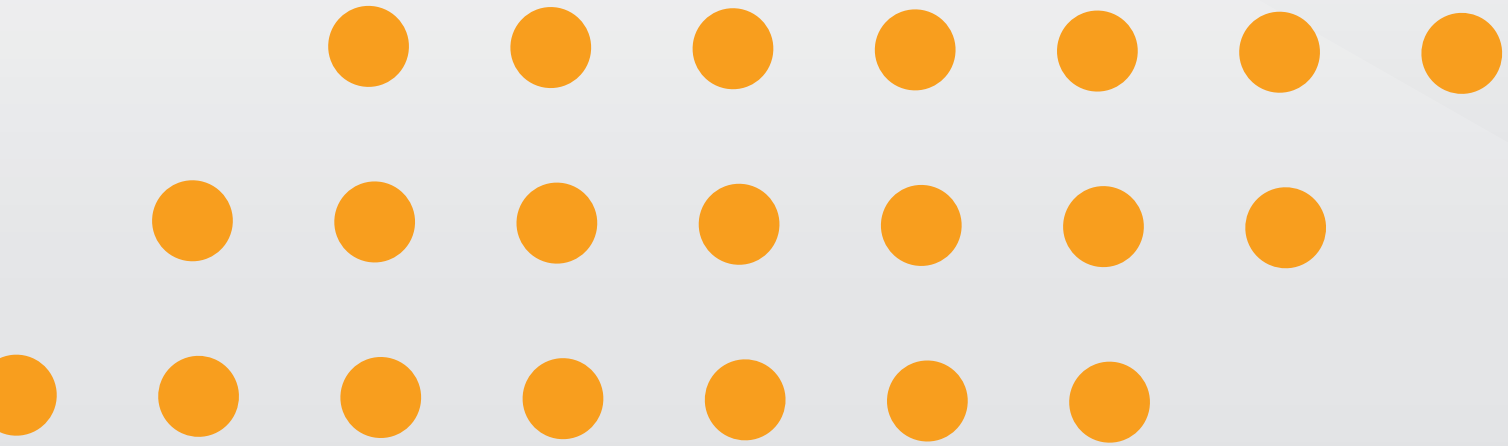
Having described all the "continuous activities" in DevOps, the following is a simple DevOps pipeline that incorporates all these activities. In its simplest form, a DevOps pipeline can have the following stages:

Plan > Code > Build > Staging > Deploy

The planning stage in DevOps is a pre-CI/CD stage where requirements are gathered, tested, and polished. Once a set of well-tested requirements is agreed upon, developers start coding those requirements. As they code, developers commit their changes to a source control repository. Each developer's source code must be accompanied by successfully executed unit tests. The commits trigger a build. Once all the code compiles properly and all unit tests pass, the build is considered successful. If any tests fail, the build is unsuccessful and the commit will roll back to a previous successful commit. This cycle repeats for the next commit.

This continuous integration process ensures that passing tests accompany every piece of code written by the developers, providing testing and code coverage at the unit level. The tested and verified build then moves to a staging environment that mimics the production environment. This is where integration tests, as well as other necessary tests, such as functional, non-functional, performance, security, user acceptance and exploratory tests are executed.

Once the application is thoroughly tested in the staging environment, it is ready to be deployed. Depending on the DevOps pipeline, it can be deployed to a pre-production environment, where the operations team may run more tests, or it can be deployed to the customers.



DevOps Testing

The way to approach testing in DevOps is to break down the pipeline into a few parts and apply different types of testing in each part. A few of those testing types are described here (this is not a comprehensive list). The simple DevOps pipeline shown above is used as a guide.

Testing during planning: This is pre-CI/CD testing. Testing during the planning stage means gathering requirements accurately and making sure they are testable by creating proper acceptance criteria. Static testing techniques, such as reviews and static analysis, should be used to ensure that defects from work products (especially requirements) are eliminated as much as possible.

Testing during coding and building: During this stage of the pipeline, unit and integration tests are written and executed.

Unit tests are developers' tests where developers are responsible for making sure that each unit being built has one or more unit tests associated with it. These tests are automated and are an essential foundation for all other tests. They are simple tests, easy to write, and fast to execute, but cover all significant paths through the code.

A common way to approach writing unit tests that ensure the testability of the code is to write the unit test first, then write the code to make the test pass, and finally doing some refactoring around that. This process is called Test-Driven Development (TDD). TDD ensures coverage around the code that is being written at the unit level. This supports continuous integration because as the developers commit their code and their unit tests, the CI system will flag any failures or missing tests. The pipeline will stop at this point until the tests pass, ensuring that only tested code will proceed.

TDD works very well at the unit level, but the tests still carry the developer's perspective. There is a need to develop tests based on other stakeholders' perspectives. Therefore, a better approach to testing during coding and building is to use Behavior-Driven Development (BDD). BDD uses the underlying concept of TDD, but instead of thinking about writing a failed test, BDD starts with writing a failed feature test. It then follows the same process as TDD to write code to pass each step of the feature test. This way, the feature is traced all the way up to a requirement.

During the coding and building stages of the DevOps pipeline, static analyzers may be used for code reviews before committing the code as part of a build. Peer reviews should also be used for reviewing the code to ensure good and consistent coding practices within the team.

Once unit testing is done at this stage, integration testing should be carried out to make sure all developer code is successfully integrated and an integration build is created. The purpose of integration testing is to ensure that there are no issues with combining the different developers' units. The integration build should pass all unit and integration tests.

Testing during staging: A critical factor to remember for testing during staging is that the environment in which testing is to take place must match the production environment. Once there is a stable build, the build can be tested in a staging environment using functional testing (does the system do "what" it should) and non- functional testing ("how well" does the system provide the functionality), such as performance, usability and other types of testing including security.

Performance testing is conducted to identify bottlenecks and any performance degradation within the system. It is done by putting demands on an application under normal and peak load conditions. It measures attributes such as response times, throughput rates, resource-utilization, and identifies the application's breaking point.

One of the main reasons that performance testing is often neglected is because performance tests

can take a long time to run and can be resource intensive. DevOps and its pipeline allow the execution of performance tests early. Some performance tests can be run in parallel with unit and integration tests. As functional tests are executed in a staging environment, performance tests for the whole system can be run in parallel.

There is a type of DevOps called DevSecOps, where cybersecurity testing is no longer considered as an afterthought but is an integral part of the DevOps pipeline. Cybersecurity testing is specialized testing and, therefore, a partnership with security experts is needed to perform such types of testing in the pipeline. In DevSecOps, security tools are identified and integrated into the DevOps toolchain and the results are made visible to the whole team.

Testing during deployment: At this stage of the DevOps pipeline, smoke tests for the whole application are performed to ensure that the application runs correctly. Every release also needs to pass acceptance tests on deployments conducted by the operations team.

The Role of Automation in DevOps Testing

Automation plays a significant role in DevOps and DevOps testing. The Agile movement embraced automation of unit testing, acceptance testing and continuous integration. DevOps ties these with automation of the deployment process, eliminating the boundary between developer and operations automation. Continuous testing would not be possible without automation; it would not be an efficient process if a large number of test cases were run manually in the DevOps pipeline. Moreover, testing is not the only place where automation is needed in DevOps.

Infrastructure as code: In DevOps, CI/CD requires the automatic deployment of changes to the test and production environments, where these environments are launched automatically as needed. CI/CD also requires an automated way to push the latest build to these environments and, eventually, to the customer's environment.

Infrastructure as code includes the process and technology needed to provision and manage environments (physical and/or virtual) through scripts.

Treating infrastructure as code is a key element in DevOps. It benefits both the development and the operations team. Infrastructure as code allows operations teams to get involved in the development process from the beginning. Developers can gain a better understanding of the supporting infrastructure because they can be involved in specifying and understanding configurations for servers, networking, storage, and so on.

The DevOps toolchain: The ultimate goal of DevOps is to streamline development with operations. DevOps does not necessarily require tools to do so, and DevOps is not defined by its

tools. However, for most organizations, tools play an important role in automating tasks and ensuring processes run as efficiently as possible.

A DevOps toolchain is a combination of tools that help in development, integration, testing, deployment, delivery, and management throughout the SDLC in a DevOps environment.

The DevOps toolchain should include tools from different categories, such as:

- Project planning and management tools
- Requirements engineering tools
- Configuration management and provisioning tools
- Source code scanning (for quality and security) tools
- Build automation and continuous integration tools
- Functional and non-functional testing tools
- Deployment tools
- Release orchestration tools
- Application and infrastructure monitoring and performance tools

It is critical to design a DevOps toolchain that is adaptable to accommodate both changes in team preference, application architecture, quality processes and other technology shifts. In order to maintain an effective DevOps pipeline, DevOps teams should include the right choice of tools as part of their DevOps toolchain.

References

ISO/IEC/IEEE Standards

- ISO/IEC/IEEE 12207:2017
- ISO/IEC/IEEE 15288

Trademarks

The following registered trademarks and service marks are used in this document:

- AT*SQA® is a registered trademark of the Association for Testing and Software Quality Assurance

Books

[Anderson00]: Anderson, L.W. and Krathwohl, D.R. (2000) A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives, Allyn & Bacon: Boston MA, ISBN-10: 080131903X

[Firtman]: Maximiliano Firtman, "Programming the Mobile Web", O'Reilly Media;

Second Edition (April 8, 2013), ISBN-10: 1449334970

[PMBOK] Project Management Institute, "A Guide to the Project Management Body of Knowledge (PMBOK Guide) – Sixth Edition, 2017, ISBN-10: 9781628251845

Other References

The following references point to information available on the Internet. Even though these references were checked at the time of publication of this syllabus, AT*SQA cannot be held responsible if the references are not available anymore. AT*SQA is not endorsing any of these sites or their products. The references are provided as a source of information only.

https://techcrunch.com/2013/03/25/ip-oh-my-gosh-all-that-money-just-disappeared/ https://www.reuters.com/article/us-facebook-settlement/facebook-settles-lawsuit-over-

2012-ipo-for-35-million-idUSKCN1GA2JR

[NASDAQ] https://www.sec.gov/news/press-release/2013-2013-95htm

National Institute of Standards and Technology. Framework for Improving Critical Infrastructure Cybersecurity. Version 1.1. 2018. https://nvlpubs.nist.gov/nistpubs/CSWP/NIST.CSWP.04162018.pdf

National Institute of Standards and Technology. Risk Management Framework for Information Systems and Organizations: A System Life Cycle Approach for Security and Privacy. Revision 2. 2018. https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-37r2.pdf

[WCAG] https://www.w3.org/WAI/policies/

www.atsqa.org