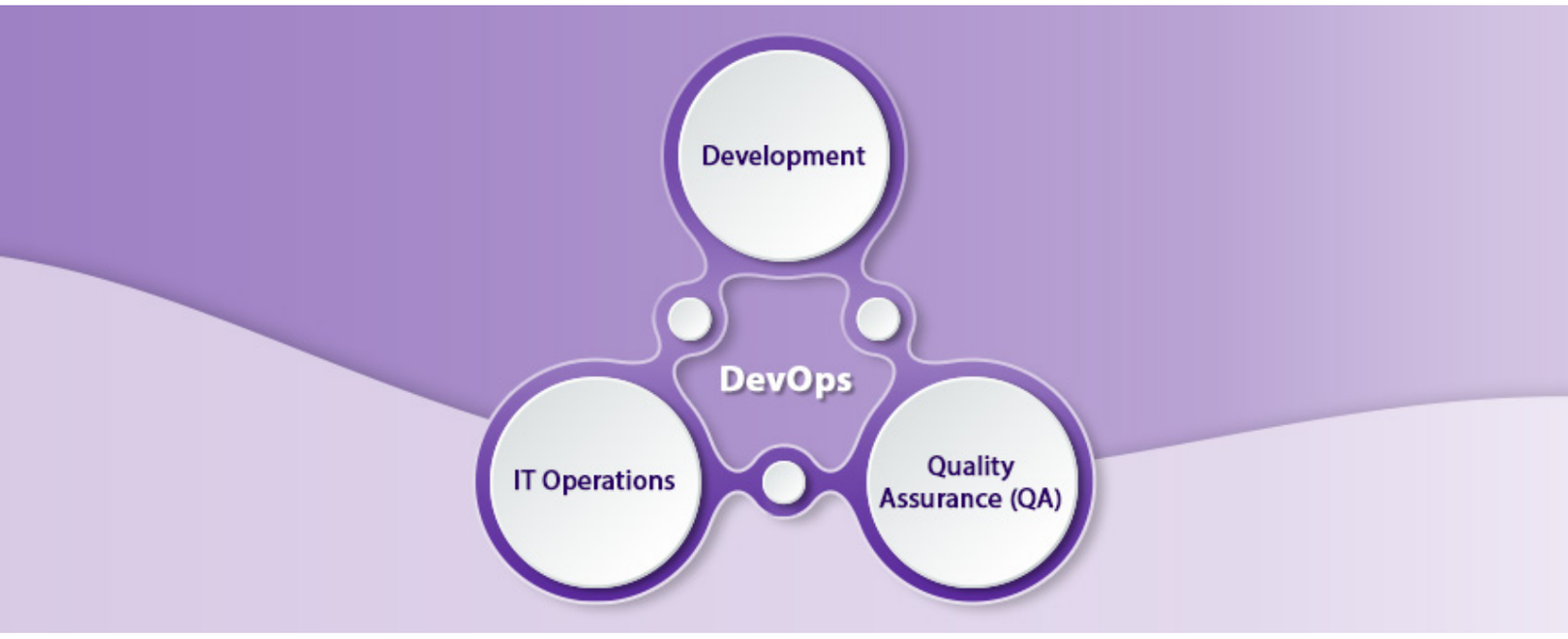
Short Guide to Successful DevOps Implementation

The term DevOps was created from two words: development (Dev) and operations (Ops). It defines an approach to software creation that unities development, testing, and operations specialists into one collaborative team as well as ensures an increase in process automation. In this blog post, we mention the advantages of DevOps and the important points of DevOps implementation strategy.

Benefits of DevOps Implementation

DevOps is gaining momentum, being an effective way to improve and accelerate the delivery of software/app/app's new features to the end-user or customer. Since the DevOps approach enhances traditional communication between in-house team members and implies complex automation of as many processes as possible, it also brings the following benefits:

- Short time-to-market
- Fewer mistakes and better code quality
- Improved teamwork and understanding of processes
- Significant cost-savings
- Simple and efficient upgrade of the product
- Accelerated debugging and recovery



The Stages of the DevOps Implementation Plan

It is always better to start the DevOps implementation process within a relatively small release cycle in order to minimize risks and check out the efficiency of such an approach. In case of success, it will be easy to scale up and use DevOps for some larger projects. Let us now highlight the main DevOps implementation steps.

1. Create a DevOps implementation strategy

Just like in all the development processes, a lot should be done before the start. Here are the things we can highlight:

Business analysis

Analyzing the market and preparing the functional requirements for future software.

Branching strategy

Choosing the way of selecting a code version for release — the branching strategy.

Organizing the team

In our opinion, the main challenge in DevOps implementation is shifting the existing corporate culture — the one with separate teams working on their specific tasks — to a collaborative environment where everyone understands and appreciates the influence of each department, to introduce a motivation system and team building events. All the team members should be open to constant communication as well as learning and utilizing new tools and solutions.

2. Start of the development process

It's a high time when software engineers join the development process. The next steps involve meetings and the adoption of new tools.

Architecture and tools

First, there is a need for creating the app's architecture based on the functional requirements as well as choosing the environment resources that will be used: databases, cache systems, messaging systems, third-party libraries, etc. It is also time to create a disaster recovery strategy and select all the other tools that are going to be utilized: from the framework for smoke tests (like Cucumber or Selenium) to CI/CD system.

Test-driven development

Start coding using the TDD (test-driven development) approach. This methodology implies that firstly the team creates tests in accordance with the functional requirements and only after that it creates a code that will manage to pass those tests.

3. Automated environment provisioning

The task here is to develop a program or a script that will automatically create and configure all the required resources for all the environments: from a 'sandbox' (an isolated environment where each engineer can test whatever wants without affecting the rest of the program) to production.

4. Set up CI (continuous integration)

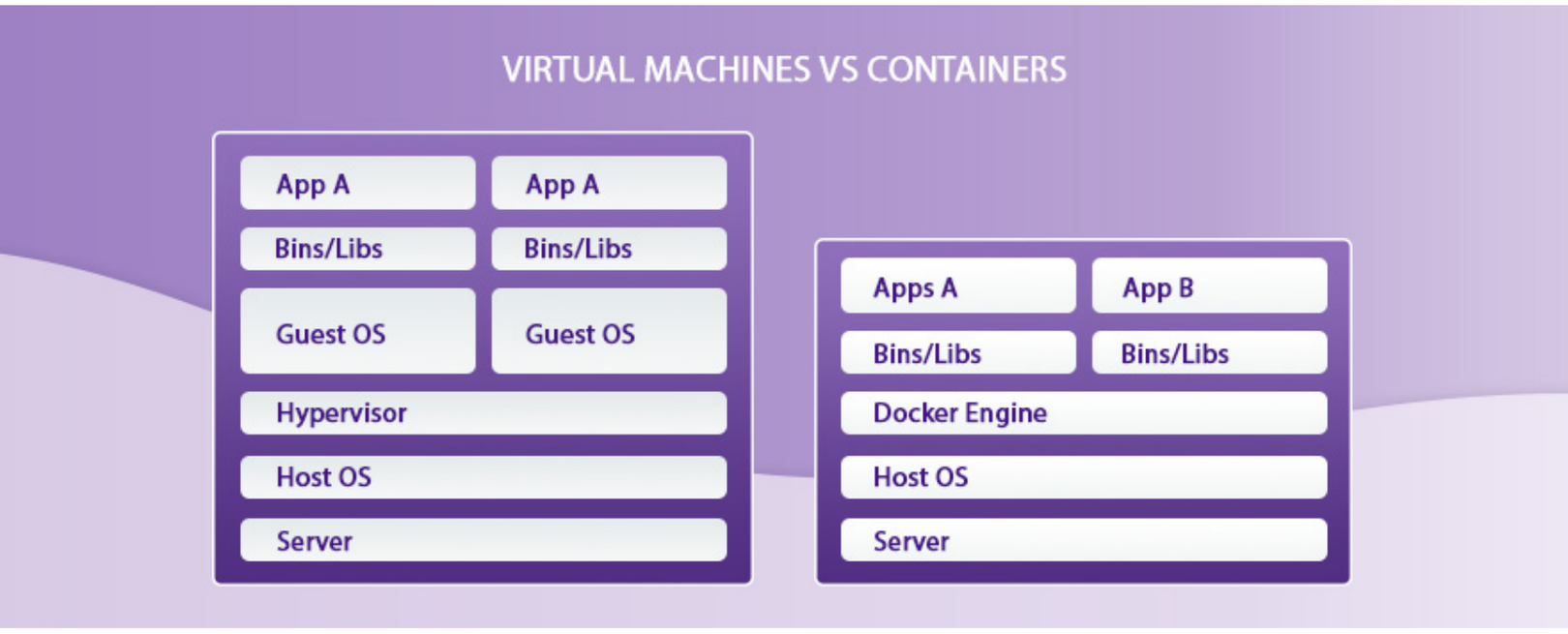
Now we have to configure the continuous integration system that will automatically compile and test the code each time a new change is committed. If a new build is verified, the system will let this code to the master branch. Here are the examples of tools you can use: Jenkins, Bitbucket Pipelines, Bamboo, TeamCity, etc.

5. Set up CD (continuous delivery)

The CD is inextricably linked to CI and responses for automatically pushing the tests-approved and verified versions of code to the next environment: from development to testing, stage, and production.

6. Containerization

Traditionally, the app is deployed either on a dedicated server or on a VM (virtual machine). This means that scalability and other changes will require significant effort, so more and more often DevOps engineers use containerization — the approach that simplifies the process of putting the build into the new environment as well as making changes. Carrying your code and all its dependencies, the container allows the app to run smoothly and reliably when moved between different computing environments. Here is the scheme illustrating the difference between using virtual machines and containers.



One more significant benefit is that different parts of a software (like front-end or database) are in several containers, so, it is easier to make changes if needed without rebuilding the whole app. The main tool here is Docker but there are also alternatives like Apache Mesos or VirtualBox. In order to manage, scale, and deploy the containers effectively, you also need a system like Kubernetes, Puppet, or Nomad.

7. Test automation and continuous deployment

After the code goes through all the previous stages, it should undergo final tests (like those emulating users' behavior from UI and making reports). All of these tests have to be automated, well-thought, and regularly updated in order to assure quality.When all the tests are successfully completed, the build is pushed to production. Now there are two scenarios:

- The code waits for being approved by someone authorized.
- The code goes straight to production.

The latest option is possible when continuous deployment is set up. It is responsible for pushing the tested build straight to production without manual verification. So, there is no need for human participation in deployment if the latest commit successfully passed all the tests. In order to entrust the deployment to the software, you have to be sure it is top-notch. It means the best possible quality of tests is a must. Such a system allows releasing updates fast enough to adopt the newest trends and to make improvements after getting users' feedback.

Blue-green deployment

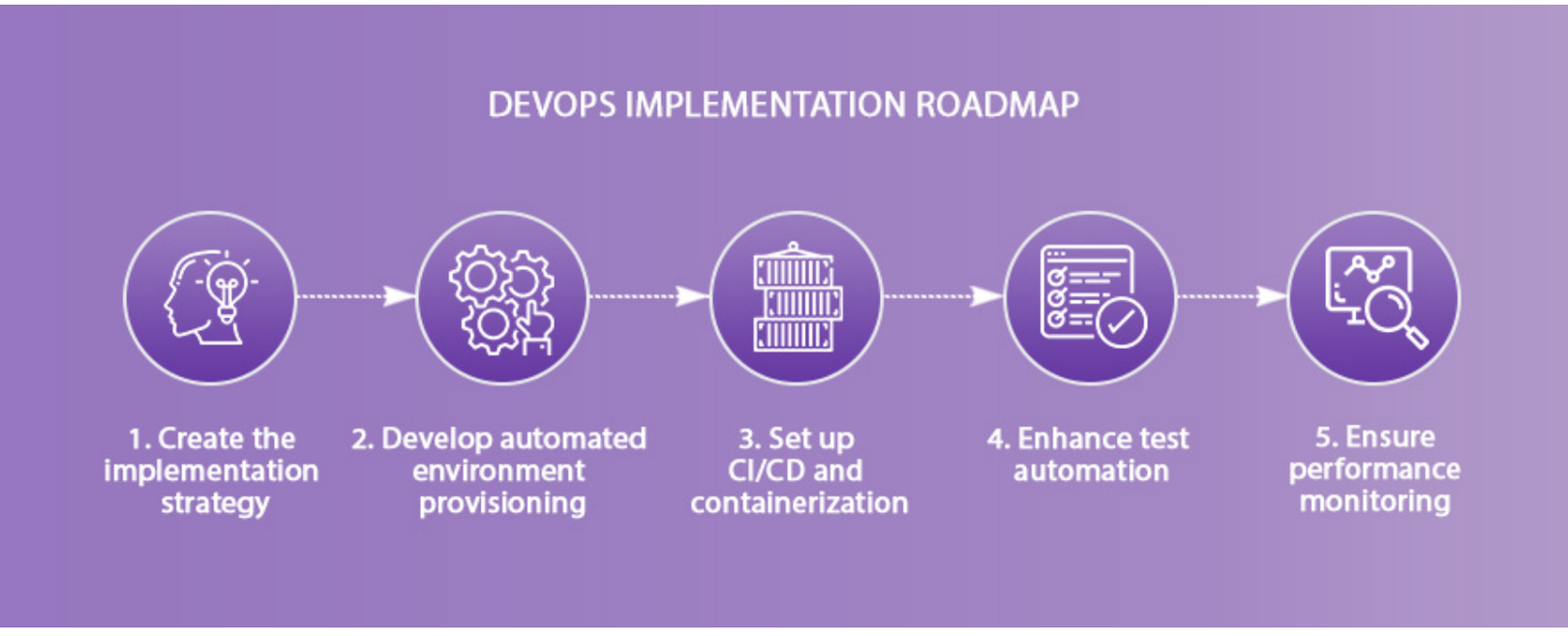
There is one more deployment strategy we want to mention here — the blue-green one. It implies having two production environments — blue and green — and switching between them with every release. So, one of them is always live while the other one is idle and can be used for testing or disaster recovery.

8. Performance monitoring

Even utilizing robust monitoring tools, it is difficult to manage and monitor everything, so the most important metrics should be highlighted, ex., detecting vulnerabilities, application performance, monitoring server health. This ensures fast notifications on any issues, hence, fixing them quickly. The monitoring tools also provide a large amount of data for analysis helping to improve the app, add useful features, scale up the infrastructure if needed, etc.

DevOps Implementation Checklist

To sum up, here is an illustration of the DevOps implementation roadmap.



Bottom Line

Just like everything that makes processes simpler and faster without compromising on quality, DevOps will keep gaining popularity. If you want to get its benefits as well as to avoid the challenges implementing DevOps, SCAND is here to help. Here is our DevOps implementation case study for review. Contact us to meet our strong DevOps team and to start a new project together!

Feel free to contact us and ask any questions: info@scand.com