DevOps Process

Group D

ENSE 375

Professor Mohamed El-Darieby

February 1, 2021

University of Regina

Jacob Chapman - 2003866001
Nolan Flegel - 200250037
Krupalkumar Patel - 200385434
Renz Rivero - 200377174
Shane Toma - 200396087

DevOps Process

Reasons to follow such a process

A solid release engineering process is important for a number of reasons. Production benefits from increased productivity while marketing benefits from distributing a higher quality product. When following a good release engineering process, the benefits amplify as the process is improved iteratively.

On a production level, a release engineering process should be followed for quality control, security, and efficient work-flow. By following a recursive test and release pattern, code is subject to error checking both by automated scripts and quality assurance personnelle. It is much more expensive to fix a bug in maintenance than it is in production, this means effective error catching will decrease the average cost of a bug. A process such as Firefox's that puts emphasis on security will largely increase the level of security of the product, it is easier to anticipate issues than it is to deal with failures. Firefox does this by planning each release as if it contains security vulnerabilities that will need to be addressed ASAP. Planning for potential security threats towards a product and instilling security redundancies is easier and cheaper than dealing with a malicious hacker. When there is a well defined system with well defined responsibilities work-flow becomes more efficient since there is a clearly laid out path for a piece of work to go whether it is functional, broken, or somewhere in between. Turnover also becomes more manageable on account of the concrete workflow system, there is little guesswork or misinterpretation possible. The clarity of workflow reduces miscommunication and confusion. Production clearly sees the bulk of the direct benefits of good release engineering.

From a marketing standpoint there are also advantages derived from the benefits of an effective release engineering system. The increased efficiency results in faster/more frequent releases, bug fixes, and feature implementation. More frequent releases enables the product to better compete in an evolving market where demands can change day by day. Better quality released products can lead to wider adoption of the software. Increased security leads to higher rates of customer satisfaction and lower potential for lawsuits. Release engineering clearly brings to the table more than just production benefits.

Improving the level of automation in a release engineering process brings a host of benefits to the developer. Errors, cost, and delays can all be reduced by implementing effective automation wherever possible. A task being automatically performed does not allow for human error and is always completed on schedule, since errors take time and time is money effective automation can also help the bottom line. When adopting a release engineering process there are not only the immediate benefits but also the potential future benefits from iterative improvements.

A solid release engineering process is important, one should be followed for the immediate benefits as well as the future benefits from improvements. Both technical and sales teams gain from release engineering. Production costs less and is more productive while marketing gets a more competitive product and decreased liability risk. The benefits experienced increase over time helping a

company remain competitive in difficult markets. At the end of the day an organization is interested in earning the best return on time and investment, by virtue of this a release engineering process is essential.

Members involved in the process

The Firefox release process is a well-developed and thoughtful piece of engineering which comprises several processes which are done with the help of automation and human involvement. The goal of this process was to make code releases simple for humans, resulting in fewer errors. Human errors can be costly, the Firefox method can help solve this issue. The release process starts from a single person known as a Release Coordinator. This person is involved with all the major departments, from the developers to the end user testers. To make the process simple, the Release Coordinator is the only person who is responsible for communicating the go or no go message to everyone. Firefox decided that this communication should only be in the form of emails. It was found that verbal and chat communication resulted in misunderstanding and problems. Therefore when a decision has been made, the Release Coordinator will send an email to a mailing list with all groups involved with the build process. It is also the responsibility of the Release Coordinator to review and sign the changes made by any team before sending out the go to build email. Automated signing has replaced the manual signing previously done by the Release Coordinator for the released builds. Automation has also saved about four to five hours of the time of updating the partial locales and platforms.

Developers are responsible for coding and creating the build packages, which is then translated into locales by an automation process. There are more than 80 of these locales that are generated. Mozilla's localization team consists of several people who are responsible for the localization of the project. Being open source there are several volunteers which are involved in this process including the internal developers. This group is also responsible for showing which changeset will be taking place, which is then displayed onto a dashboard. The next stage involves the Quality Assurance team.

Quality Assurance is the process where contractors and other community members play their role of testing whether the software update is performing the tasks correctly. They are responsible for verifying the program's operation with manual testing. After the QA team has signed off on the build, the Release Coordinator is responsible for final checks before pushing the build for release to the end users. This is accomplished by sending the go to build email where the automated build and release process is initiated.

The PR and Marketing department oversee where the consumer has access to the software package. They remind the existing customers of the new product information as well as attracting new customers to use the product.

DevOp Tools

Docker

Docker is a software platform that allows developers to package their code into units called containers. These software containers are self sufficient and isolated from the system they are running on. They can be copied, transported, shared and run on virtually any platform. Docker is a software service similar to a hypervisor platform, the system virtualizes a hardware system for the containers to use. The advantage of Docker over virtual machines is each container is that docker shares operating system kernel and resources with each container, allowing the containers to remain small, lightweight and efficient. Individual containers isolate their system kernel and remain abstracted from the host system and other containers. Each container can have its own subset of packages, libraries and components that are maintained separately from other containers on the same system. This allows software to be developed, tested and deployed independently and in isolation from other services running on the same system. The Docker system reduces the overall system overhead needed to run multiple applications and allows the host system to be more efficient and run more applications than other hypervisors. Containers and microservices are popular methods of software delivery on cloud services platforms like Azure, the containers are virtualized and easily scaled within these cloud environments.

Azure DevOps

Azure DevOps is one of the services provided by Microsoft's Azure Cloud Computing Platform. Azure offers hundreds of different services and infrastructure available for businesses and individuals. The Azure cloud platform is not exclusively for Microsoft software, Linux and other platforms are also supported. Azure DevOps is a Software as a Service (SaaS) platform that provides tools to developers that aid in development and deployment of software applications. This service integrates with other popular development tools to provide a comprehensive solution that covers the full development life-cycle for a software application. There are services available for agile planning, repository hosting, package management, software container management, Continuous Integration and Continuous Deployment and more. These services are platform agnostic and can be utilized by any organization or workflow. The Azure DevOps platform is focused on providing regular updates and access to the latest features and tools and scalability. The cloud platform is available globally and offers an SLA of 99.9% uptime. By maintaining the DevOps toolchain for developers, the SaaS model allows developers to focus on development and not spend resources on maintaining their toolchains.

Continuous Integration / Continuous Deployment

Continuous integration, Continuous Delivery and Continuous Deployment known as CI/CD is a software development philosophy and practice of implementing small software changes in a consistent way. The goal of CI/CD is to implement an autonomous method of testing and deployment for the lifecycle of software applications. With Continuous Integration, development changes are automatically built, tested and integrated into a code repository. This prevents code fragmentation and disorganized

development branches and helps manage code conflicts. Continuous Delivery and Continuous Deployment are closely related to each other and often used interchangeably. The CD part of CI/CD involves further automation of the development pipeline, code changes are automatically pushed to staging environments for testing and review. The distinction between terms for Delivery and Deployment is dependent on how sophisticated the development pipeline is. When abstraction between the different environments is needed, Delivery often refers to staging/testing environments and Delivery refers to production environments. CI/CD pipelines enable developers to have more frequent software changes

Git/GitHub

Git is a free and open source version control system that is installed on a local machine and can be used to track code changes. It allows programmers to create independent branches, delete, merge and revert code changes. Git is a useful tool for developers because it allows simultaneous development of projects and automatic tracking of changes. It is a comprehensive system that can handle multiple projects, accounts and workflows.

GitHub is a cloud service owned by Microsoft that offers repository hosting services. GitHub uses Git version control to manage remote code repositories for organizations or individuals. This cloud based service offers redundancy and protection against system failure, the ability to easily collaborate, share and access repositories from anywhere.

VS Code

Visual Studio Code (VScode) is a free and open source code editor produced by Microsoft. Important features include cross platform support, built in terminal command line interface (CLI), thousands of available extensions, and embedded Git support. Cross platform support makes collaboration easier and working on different platforms a low barrier to development. The vast collection of available extensions make VScode highly customizable and versatile, by adding extensions VScode can support virtually any contemporary programming language. The built in terminal CLI grants a high level of convenient control to the programmer over their work. The embedded Git functionality makes VScode ideal for use in version control. VScode is a valuable tool for release engineering. It provides a productive and flexible coding environment and integrated source control allows development teams to connect to their CI/CD pipeline.

The Inner-loop process

Before executing the outer-loop workflow, which spans the entire devops cycle, there is a tool that is common for each developer. No matter what coding language or platform is used, the application will be running and tested on Docker containers locally.

A container / instance of a Docker image has four main components which are:

- A host / Operating System
- Files appended (e.g. application binaries)

- Configuration (e.g. application dependencies, environment variables)
- Instructions / commands to be run by Docker

Figure 1 shown below from Microsoft shows a high level process for the life cycle of Docker and the steps needed to build and deploy an application.

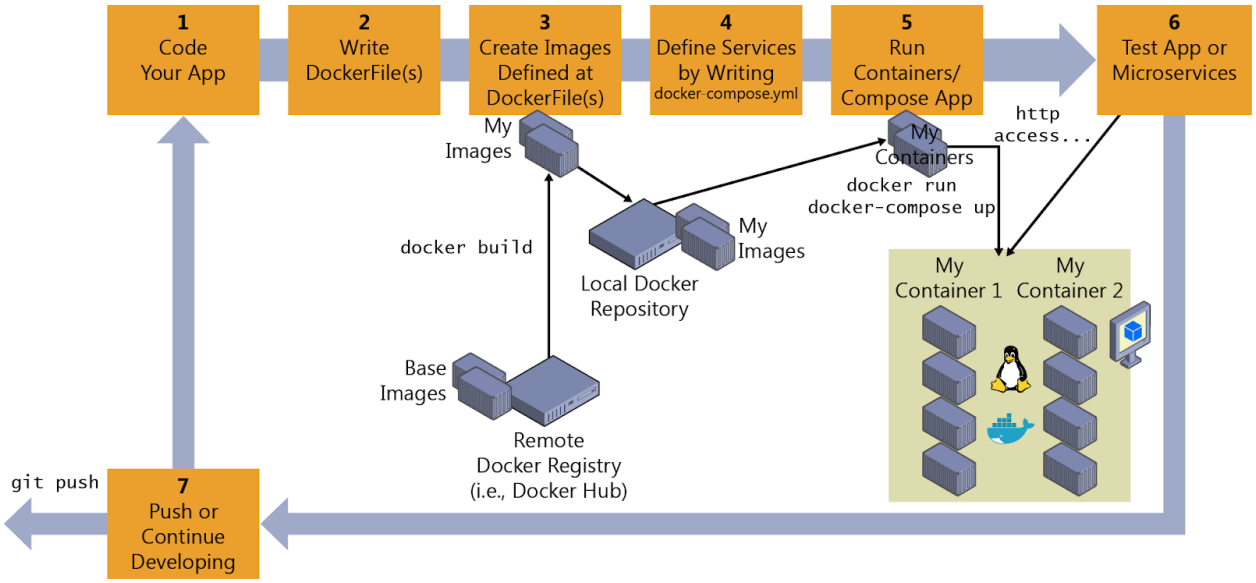


Figure 1: Docker Workflow (Microsoft 2021). From Inner-loop development workflow for Docker apps.

https://docs.microsoft.com/en-us/dotnet/architecture/containerized-lifecycle/design-develop-containerized-apps/docker-apps-inner-loop-workflow

The first step as shown in figure 1 above is to code the application. The method on how the application is developed does not change. The only difference is that while developing, the application is deployed on Docker containers that are in a local environment. Step two of the Docker workflow is to create a Dockerfile for the application. A Dockerfile is a document containing all the commands a user or developer calls in the CLI to assemble a Docker image. Each custom made image needs a separate Dockerfile to be built and deployed. If a single service is used, only one Dockerfile is needed. Otherwise, if there are multiple services, one Dockerfile per service is required.

Inside a Dockerfile, the first line of code specifies the base image that is referenced. Developers then add on top of the base image, so there's no need to reinvent the wheel. Once a Dockerfile is created, developers build the image by utilizing Docker commands in the CLI. By using the 'docker build' command in the CLI, this will build an image locally. After building the image, it can be run inside a container by using the 'docker run' command. Along with a *Dockerfile*, developers can make a *docker-compose.yml* file that can define a set of related services to be deployed as a composed application. If the application only has a single container, all that is required to run is to deploy the

application to the Docker host. However, if the application has multiple services, each will need to be composed.

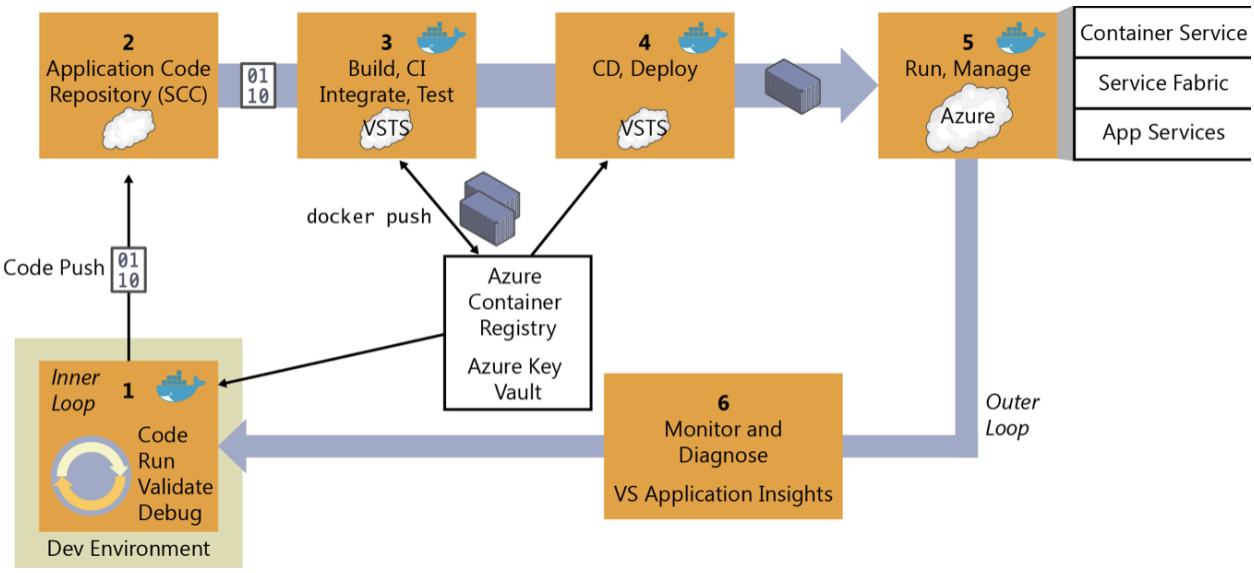
If developers have a Docker application with multiple services, they utilize the command 'docker-compose up'. This command will build custom images for each defined service inside the *docker-compose.yml* file.

In conclusion, Docker is an important tool that is widespread throughout each developer's machine. This tool allows developers to create, build, and deploy images as a single service or multiple services using the CLI. These images are created, built, and run locally on each developer's local machine and is a powerful tool that is secure, provides fast testing feedback and is a secure way to deploy applications.

The Outer-loop process

Figure 2 shown below from Microsoft shows the steps that make up the development and operations throughout the outer-loop workflow.

Figure 2: Outer-loop workflow for Docker applications (Microsoft)



Note. Microsoft (2021). *Figure 5-1*. From Steps in the outer-loop DevOps workflow for a Docker application.
https://docs.microsoft.com/en-us/dotnet/architecture/containerized-lifecycle/docker-devops-workflow/docker-application-outer-loop-devops-workflow

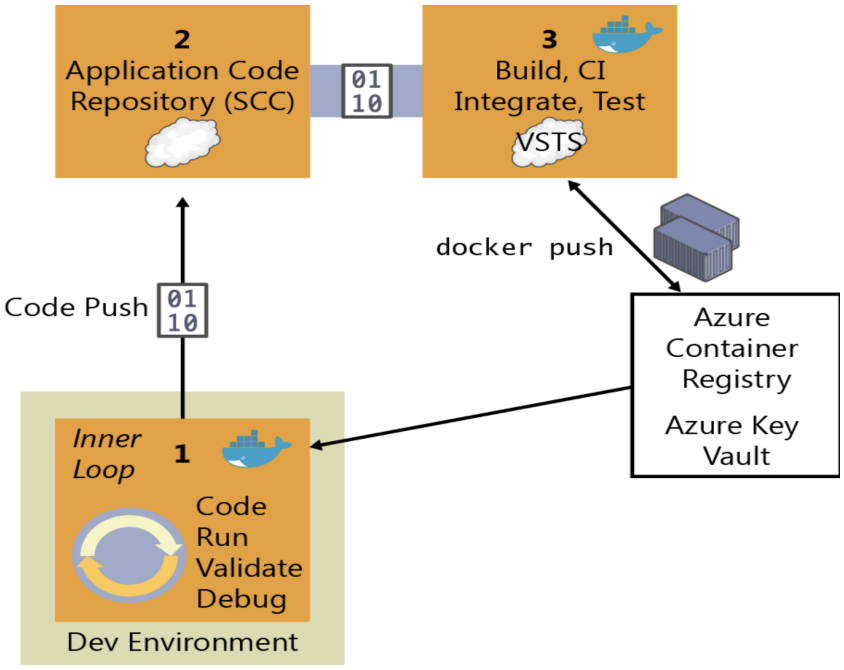
The first step as shown in figure 2 above is explained in detail in the preceding section. It is where the outer-loop starts, the moment a code is pushed to a repo, a CI pipeline is initiated to perform actions. The second step for the outer-loop workflow for Docker is to have a version control system like Git/Github to collect and combine versions of all the code from different developers in a team. When

creating Docker applications in a software development and IT operations life cycle, it is crucial that a developer must not submit the Docker images with the application directly to the main global Docker Registry unless these Docker images to to be released to production environments are created on the source code being integrated in the main global build or CI pipeline based on the source-code repository like Git/Github.

The local Docker images generated by developers are for their own use when it comes to testing in their own machines. It is why it is crucial to have the software development and operations pipeline activated based on the source-code.

Once a version-control system is determined with the correct source code submitted, a build service is then needed to grab the code and run it on the global build and tests. The internal workflow for this step as shown in figure 3 below is about building a CI pipeline consisting of the source-code repository (Git/GitHub), build server (Azure DevOp Services) , Docker Engine, and a Docker registry.

Figure 3: The steps involved in Control Integration (CI)



Note. Microsoft (2021). *Figure 5-2*. From Steps in the outer-loop DevOps workflow for a Docker application.
https://docs.microsoft.com/en-us/dotnet/architecture/containerized-lifecycle/docker-devops-workflow/docker-application-outer-loop-devops-workflow

The basic CI workflow steps with Docker and Azure DevOps Services:

- A commit/push to a source-code control repository initiates a CI pipeline

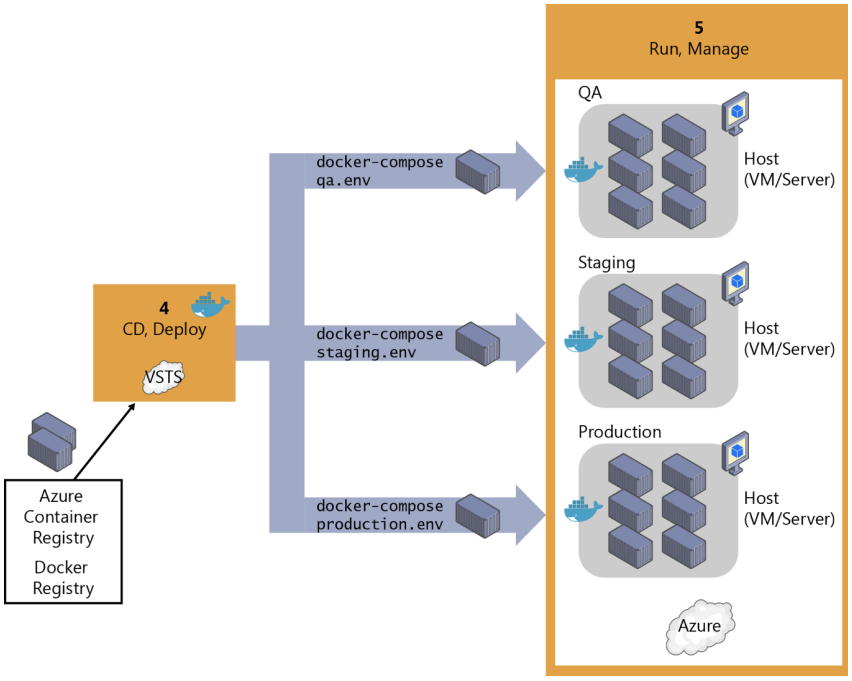
- The commit/push causes Azure DevOp services to grab a Docker container and run a build job
- On successful run of that built job, a Docker image is pushed to the Docker Registry

The next step of the process after having the Docker images published in a Docker Registry is to deploy them onto several environments from the CD pipeline with the use of the Azure DevOps Services. At this point, depending on the kind of Docker application to be deployed, two scenarios are produced:

Deploying a composed Docker application to multiple Docker Environments

In the scenario of deploying a simple application making up of a few containers or services to be deployed to a few servers, the internal CD pipeline can use docker-compose through the Azure DevOps Services to deploy the Docker applications with its related set of containers and services. This is shown in figure 4 below.

Figure 4: Deployment of application containers to Docker host registries



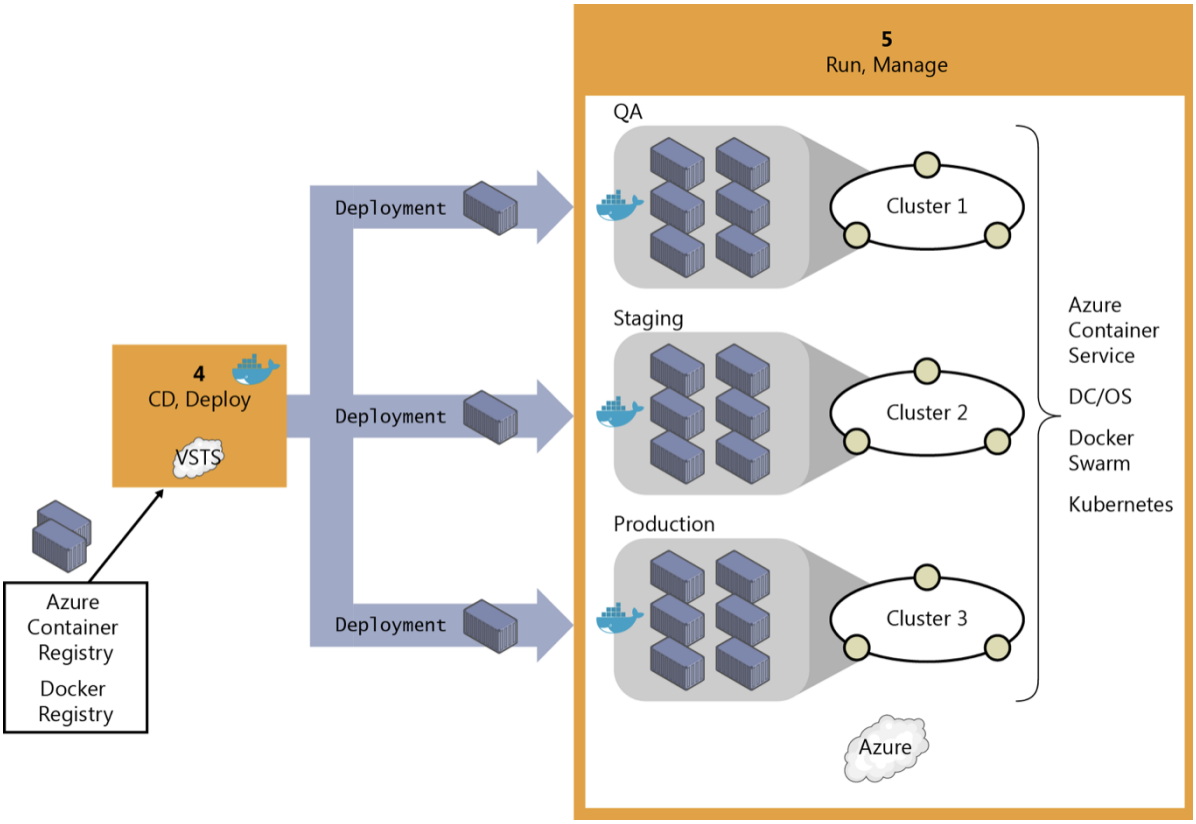
Note. Microsoft (2021). *Figure 5-2*. From Steps in the outer-loop DevOps workflow for a Docker application.
https://docs.microsoft.com/en-us/dotnet/architecture/containerized-lifecycle/docker-devops-workflow/docker-application-outer-loop-devops-workflow

Deploying Docker applications to Docker Clusters

A more complicated scenario in which distributed applications require compute resources that are also distributed. In order to have scaling capabilities in production, pooled resources need to have clustering capabilities that provide high scalability and availability.

A much complicated scenario in which, from a continuous delivery point of view, can run specially made deployment tasks from Azure DevOps Services that will deploy the Docker containerized application to a dispersed cluster in a Container service. This is shown in figure 5 below.

Figure 5: Deployment of distributed applications to a Container Service



Note. Microsoft (2021). *Figure 5-2*. From Steps in the outer-loop DevOps workflow for a Docker application. https://docs.microsoft.com/en-us/dotnet/architecture/containerized-lifecycle/docker-devops-workflow/docker-application-outer-loop-devops-workflow

As for the final steps, it is all about running and managing the applications at the enterprise-production level, performing load testing, and monitoring beta or QA environments.

References

Anil, N. A., Ghosh, S. G., M., Kulikov, P. K., Victor, Y. V., Wenzel, M. W., & Parente, J. P. (2021, January 6). *Steps in the outer-loop DevOps workflow for a Docker application*. Microsoft Docs. https://docs.microsoft.com/en-us/dotnet/architecture/containerized-lifecycle/docker-devops-workflow/docker-application-outer-loop-devops-workflow

Anil, N. A., Ghosh, S. G., Thalman, M. T., Schonning, N. S., Kulikov, P. K., Victor, Y. V., Wenzel, M. W., & Parente, J. P. (2021, January 6). *Inner-loop development workflow for Docker apps*. Microsoft Docs. https://docs.microsoft.com/en-us/dotnet/architecture/containerized-lifecycle/design-develop-containerized-apps/docker-apps-inner-loop-workflow

Atlee, C. A., Blakk, L. K., O'Duinn, J. O., & Gasparian, A. G. (n.d.). *The Architecture of Open Source Applications (Volume 2): Firefox Release Engineering*. The Architecture of Open Source Applications. http://aosabook.org/en/ffreleng.html