Continual learning on deployment pipelines for Machine Learning Systems

Qiang Li[*]
Accenture
Kaistraße 20, 40221 Düsseldorf
`qiang.i.li@accenture.com`

Chongyu Zhang
Technical University of Munich
Arcisstraße 21, 80333 München
chongyu.zhang@tum.de

Abstract

Following the development of digitization, a growing number of large Original Equipment Manufacturers (OEMs) are adapting computer vision or natural language processing in a wide range of applications such as anomaly detection and quality inspection in plants. Deployment of such system is becoming a extremely important topics. Our work starts with the least-automated deployment technologies of machine learning systems, includes several iterations of updates, and ends with a comparison of automated deployment techniques. The objective is, on the one hand, to compare the advantages and disadvantages of various technologies in theory and practice, so as to facilitate later adopters to avoid making the generalised mistakes when implementing actual use-cases, and thereby choose a better strategy for their own enterprises. On the other hand, to raise the awareness of the evaluation framework for the deployment of machine learning systems, to have more comprehensive and useful evaluation metrics (e.g. table 2), rather than only focusing on a single factor (e.g. company's cost). This is especially important for decision-makers in the industry.

1   Introduction

Increasingly, large-scale machine learning systems including front and back end, cloud platforms, and integrated ecosystems of computer vision or natural language processing are being deployed and leveraged with great value. Meanwhile, a variety of state-of-the-art models and the pursuit of innovative techniques in academia are in turn facilitating this digital revolution. However, we can note that the discussions on a specific technology are far more heated than the discussions on the need to use that technology (1), and it is unfortunate that with the trend of KPIs in the industry, we tend to miss the fairness of evaluating deployment technologies from a more diverse perspective.

It is worth noting that all the experiments and conclusions in this paper have been tested in industry-specific cases, and this paper can be used both as a technical handbook for the deployment of ML System (e.g. quality inspection systems), also as a valuable reference for the selection of different technologies.

2   Methodology

2.1   How Edge Device and NSSM Batch Services enable Static Deployment

For most industrial applications, it is reasonable to build an MVP demo and implement some well-known algorithms for quality inspection systems, such as ResNet (2) or Fast-RCNN (3), based on
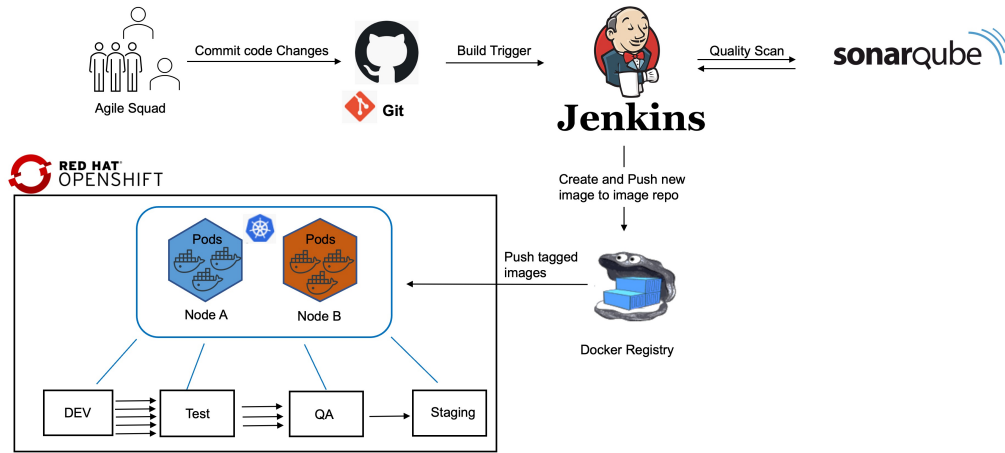

Figure 1: Jenkins and OpenShift CI/CD Pipeline (5). First upload the model to the cloud (e.g., model directory), then update the business logic code by git; Jenkins acts here as a monitor and can be triggered to build the docker image. Then the docker image name and the tag will be forwarded to the OS (9), and the AI container will be saved in Nexus. Finally, the OS can use the latest docker image build to complete the deployment. Only the settings file about the application needs to be set up here, where we can very easily define the speed of sending images, the processing time for each inference, and the CPU/resources for each use case.

well-established TensorFlow Object Detection API (4). Without the support of the cloud platform, through NSSM (7) and Edge device, it is also possible to quickly achieve the deployment and construction. You may doubt the stability of those systems. On the contrary, it is often possible to achieve a real industrial use-case of recognition with a significant amount of data volume (e.g., 3 T image data per day) through these tools. Figure 3 and Figure 4 describe the specific implementation of NSSM Batch Service-based Deployment in great detail.

Although it is stable and appropriate for the initial stage of the product, this deployment technology also reveals some weaknesses, such as the complex deployment steps that often bother some senior AI engineers. The second is very poor scalability, and if the model goes wrong, this often does not achieve automatic deployment, demanding killing or suspending the previous batch service. The port connectivity issues, how to use it, and which port to use require much attention. In addition, due to the existence of edge devices, we often need a third-party virtual machine connection tool like NetSupport Manager (see Figure 3). When deploying large-scale machine learning systems, with the increasing complexity of use cases and the cooperation of the various regions, this deployment method of installing a separate edge device and NSSM batch service is often not recommended (7).

2.2 Semi-Automatic Deployment using Jenkins and OpenShift

The invention of Jenkins (5) makes the deployment of models much easier, and of course, Jenkins is not the only technology utilized here. The above Figure 1 nicely illustrates the entire deployment process from docker generation, uploading to the Openshift container storage, and finally, the trigger and scaling process.

As shown in Figure 6, many Jenkins operations are graphical, making the deployment process more intuitive. The user only needs to click on some build buttons to trigger docker image builds. However, this deployment method also has a significant drawback, namely the accessibility of the logs. In some cases, it is unclear which part of the deployment process has failed due to the limited length of the log. Furthermore, it is deplorable that the docker image is directly connected to OpenShift, so getting the log and the docker image for a local test is not very straightforward. It is frequently

2

only by the end of the deployment that we can tell if the image is a successful docker image by the successful running of the final application from Openshift. A successful docker image (see Figure 9), should contain all the necessary library packages and the inspection part, including the business post-processing logic when the image passes through the dispatcher API.

Log4j security concerns   It is also worth paying attention to the security of Jenkins-based semi-automatic deployment methods, such as log4j security concerns caused by Java (6).

2.3   Run In Fly: OpenShift VS Kubernetes

OpenShift is a cloud development Platform as a Service (PaaS) developed by Red Hat, enabling developers to develop and deploy their applications on cloud infrastructure (9). It is a well-established platform and relatively straightforward for AI engineers to get started and deploy an application based on its learning cost compared to other PaaS platforms (see in Table 2). There are many benefits adopting Openshift-based PaaS services in real industrial applications, such as graphical operations. AI developers can get on board quickly, even if unfamiliar with the backend architecture since backend support is available from this platform as soon as the docker image is uploaded successfully.

OOM killed: resource concerns   Besides, this approach has also many shortcomings. It is often the case that after deployment, due to resource issues, OOM (Out of memory killed), frequently occurs for a single use case (see in Figure 7. Poor scalability is also a critical issue when resources become very expensive while simultaneously needing to process a massive amount of data. Limited storage and CPU resources of OpenShift often make people give up on this platform.

Transparency concerns   The transparency is a topic of great concern to computer vision algorithm engineers as well. Engineers often need to index logs, while Openshift often has only a limited number of inspection logs and often connects to Kibana (12) ; it takes much time to query these columns. Meanwhile, OpenShift reserved many useless logs for network layer ping pang, which accelerated resource consumption while increasing costs for the enterprise.

2.4   Cloud Migration bring new CI/CD Pipeline

To retain the benefits of OpenShift deployments, it may be a good choice to go with a more cost-effective cloud provider, but cloud migration also brings a lot of new tools and challenges, such as K9s, or the ArgoCD-based deployment tools that follow (13). More obviously, this approach brings more testing in terms of docker image. This feature allows AI engineers to realistically check the success of a docker image build. (12)

2.5   How ArgoCD leverage the Deployment

As shown in the Figure 8, ArgoCD is also a Kubernetes-based deep learning application deployment tool and controller. Similar to Jenkins, they both have the same automation feature where any changes made to the desired target state in the Git repository can be automatically applied and reflected in the specified target environment. Unlike PaaS-based deployments such as Openshift, here we can monitor the server's state in real-time and easily modify or refresh, delete, and synchronize the state of Pods. With repository servers, many operations involve commits, tags, and branches, so a higher level of git knowledge is required. In addition, it is straightforward to implement local testing of docker images. Logs and CPU resources for individual pods are no longer an issue. Once the model is deployed successfully, it can run stably and will not encounter OOM problems like the Openshift platform-based deployment approach (13).

Parallelism concerns   Similarly, server and git-based deployment methods have many issues; for example, one of the highlights of this deployment method is the need for the tag, which requires many git operations, so if the team is deploying multiple use cases at the same time, parallelism will become very problematic, which requires a dedicated person to control on git merge operations. The consequence is that each deployment takes twice as long as the previous Openshift-based methods. Since all CPU parameters of the model are controlled by some configuration, each modification of these parameters requires a new git operation, so it becomes very cumbersome for the rescaling and redeployment process.

2.6 Automatic Deployment based on GitHub, Docker and ArgoCD

Whether based on Openshift or ArgoCD, the model deployment process requires more or less manual effort to build the docker image through a series of command lines. Assuming a wrong step on the command line, or a wrong version number of a python package, these will directly lead to the failure of the docker image, and the process of building docker requires the installation of all the environment files and packages, which is no less than installing a single software. General principles dictate that the docker image is the sandbox that allows the AI model to run successfully. How to automate or semi-automate this process can be done using ArgoCD, Helm, and Git. The diagram below Figure 10 gives a decent overview of this process.

With the above approach, we automate the generation of docker images, but this can also bring up issues of low transparency and difficulty in local testing.

3 Challenges and Evaluation Metrics

Learning cost for engineers: This is directly related to the educated-market of this deployment method. If it is a very mature company with operating system support like PaaS, it may be speedy for deployment and easy to learn, which is necessary for enterprises to build their initial machine learning system.

Transparency: of a deployment technique or deployment system will directly determine the speed of debugging and later find the cause of model problems. Academia is very enthusiastic about the accuracy of artificial intelligence, but less popular papers can be seen on the interpretability and stability of models (1). In industrial applications, we often encounter situations where models perform very well in training and testing, but face problems once deployed - so transparency in all of this should be important, starting with model training, dataset selection, business code for edge cases, transparency of docker images and accessibility of deployment logs.

Stability: for example, when the model is deployed successfully, the model should be able to run relatively stable. Here, particular emphasis is placed on the Stability of the machine learning system deployment pipeline. Different deployment methods have many advantages and disadvantages, from static to semi-automatic and from static to automatic (see in table 2). Moreover, if the enterprise often changes the deployment method for reasons such as saving economic costs, it will cause the consequences of system instability. Because machine learning engineers need to spend time and effort to test and learn new deployment methods and migration, this is not conducive to improving model stability, which can also directly lead to delays in product delivery.

Parallelism: for machine learning deployment, large teams often have very high requirements for parallel deployment, where models, docker images, and deployment files should remain independent and uncorrelated (14). High independence can ensure that the source of AI problems can be found quickly during the debugging process, and high Parallelism can ensure the rapid deployment of AI models and, thus, the rapid delivery of products.

Cost: the choice of deployment method often depends on the cost of this deployment method, which is one of the most important reasons for the enterprise to choose this method and the core reason for going through many iterations. However, for PaaS-based deployment methods such as OpenShift (9)(11), the price of different cloud platforms should not be the ultimate measure. The wisest approach should be to choose the most suitable solution after measuring the scale of the current machine learning system; otherwise, frequent changes on deployment methods or cloud platform providers in pursuit of the cheapest solution can also lead to unstable machine learning systems.

4 Conclusion

This paper discusses the advantages and disadvantages of different deployment methods and the challenges encountered from the perspective of the practical large-scale industrial machine learning system. We propose the metrics and strategies that should also be measured when choosing a deployment method. In this paper, we also discussed some current misconceptions and common mistakes made from the perspective of engineers and stakeholders. This paper can be a good reference for choosing a suitable deployment method for Production ML Systems and a handbook for debugging reference schemes.
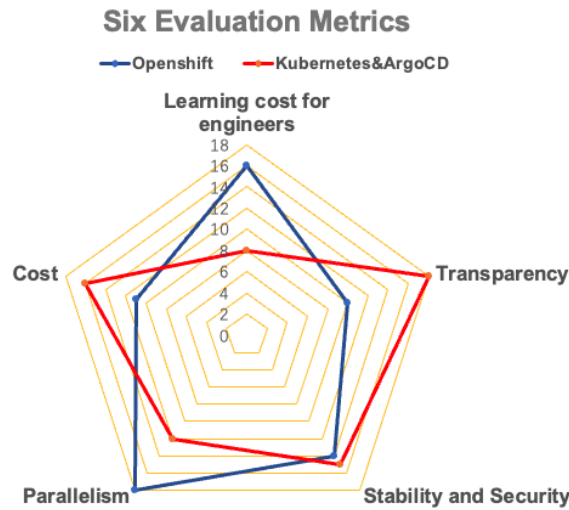
References

[1] Wiemer, H., Drowatzky, L. & Ihlenfeldt, S. (2019) Data mining methodology for engineering applications (DMME)—A holistic extension to the CRISP-DM model. *Applied Sciences* **9**(12): 2407.

[2] He, K., Zhang, X., Ren, S. & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770-778.

[3] Ren, S., He, K., Girshick, R. & Sun, J. (2015). Faster r-cnn: Towards real-time object detection with region proposal networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama and R. Garnett (eds.), *NIPS*, pp. 91-99.

[4] Vladimirov, L. (2020) TensorFlow 2 Object Detection API tutorial. Retrieved from `https://tensorflow-object-detection-api-tutorial.readthedocs.io/en/latest/`.

[5] Beck, D. (2022) Jenkins User Documentation. Retrieved from `https://www.jenkins.io/doc/tutorials/#pipeline/`.

[6] Yazıcı, V. (2022) Apache Log4j Project Guidelines. Retrieved from `https://logging.apache.org/log4j/2.x/guidelines.html`.

[7] Patterson, I. (2017) NSSM - the Non-Sucking Service Manager. Retrieved from `https://nssm.cc/commands`.

[8] "NetSupport" (2022) NetSupport Manager Manual – Version 14.00. *NetSupport Ltd.*, Retrieved from `https://resources.netsupportsoftware.com/resources/manualpdfs/nsm_manual_uk.pdf`.

[9] "Red Hat Openshift" (2022) OpenShift Container Platform 4.11 Documentation. Retrieved from `https://docs.openshift.com/container-platform/4.11/welcome/index.html`.

[10] Galiana, F. (2020) Kubernetes CLI To Manage Your Clusters In Style! Retrieved from `https://k9scli.io/`.

[11] Dlugy-Hegwer, R. (2022) Red Hat OpenShift Documentation. Retrieved from `https://www.redhat.com/en/technologies/cloud-computing/openshift`.

[12] Gupta, Y. (2015) Kibana essentials. pp. 89-112. Birmingham, UK: Packt Publishing Ltd..

[13] Matyushentsev, A. (2022) Argo CD - Declarative GitOps CD for Kubernetes. Retrieved from `https://argo-cd.readthedocs.io/en/stable/`.

[14] Rad, B. B., Bhatti, H. J. & Ahmadi, M. (2017) An introduction to docker and analysis of its performance. *International Journal of Computer Science and Network Security (IJCSNS)* **17**(3): 228.

A   Appendix

Table 1: General workload for machine learning System Development. As stated from (1), 80 percent of modern ML project delivery time are spend on pre-evaluation and model understanding and training occupies 20 percent. However, the right strategy for deployment and techniques are lessly discussed (1).

Business Understanding	Data Understanding	Data Preparation	Modelling	Evaluation	Deployment
Determine Business	Collect Initial Data	Select Data	Select Modeling Techniques	Evaluate Results	Plan Deployment
Assess Situation	Describe Data	Clean Data	Generate Test Design	Review Process	Plan Monitoring and Maintenance
Determine Data Mining Goals	Explore Data	Construct Data	Build Model	Determine Next Steps	Produce Final Report
Produce Project Plan	Verify Data Quality	Integrate Data	Assess Model		Review Project
		Format Data			
		Dataset			

Six Evaluation Metrics

— Openshift — Kubernetes&ArgoCD

Learning cost for engineers

Cost

Transparency

Parallelism

Stability and Security

Scores sum up to 100, the higher the better.

Metrics	Learning cost for engineers	Transparency	Stability & Security	Parallelism	Cost
100 P	The complexity of CI/CD deployment pipeline infrastructure: • Discrete development (1p) • Continuous integration (1p) • Coding language & syntax complexity (2p) Documentation tutorial: • Is this deployment pipeline well supported by the company or community? (4p) • Is this official tutorial documentation accessible? (4p) Engineer Learning time: • Can this deployment pipeline be mastered by the engineer within short time (< 2 sprints in agile development circle needed?) (8P)	Transparency in deployment logs: • Provided deployment logs but with limited info? (2p) • No logs? (0p) • Informative logs and broken deployment can be debugged easily. (4p)* Transparency in built docker image: • Is local testing docker image possible? (4p) • Is docker image building process trackable and can we see the log? (4p) Transparency in running docker image: • Is AI inference log accessible? (2p) • Is the log of running docker container informative and can be easily root tracked on the issues easily? (6p)	External Stability & Security • What kind of language is this CI/CD pipeline tool developed from, does it **NOT** have well-known pre-existing vulnerabilities? (such as log4j vulnerability by Java) (4p) • Does this CI/CD pipeline **NOT** need a lot of external third-party plug-in libraries/ API or supports? (2p) • Is it developed by a trustful company or organization, or a well-organized open-source community? (4p) Internal Stability & Security • Dependency of AI packages (1p) • Reproducibility on error/ bugs (2p) • Recoverability on the docker image (1p) • Rollback of a failed deployment (2p) • Stability of running AI model (container) (2p) • Consistency of deployment methods (2p)	Development event: • Can multiple engineers within the team submit the deployment request? (4p) • Can multiple deployment requests be handled parallelly? (4p) • Can multiple build-in docker images be created at the same time? (2p) Resources & scale • **NOT** limit the number of concurrent instances or **NOT** limited parallel running containers? (4p) • Single processing of AI inference or multiple data processing at the same time? (4p) • **Not** limited resources of CPU/ GPU power? (2p)	Project management & Deployment (10p) • Person cost • Domain cost • Tools cost • License cost (Cloud/ ATC git, platform related) • Storage cost Model & Coding & Validation (10p) • Inference time cost • Teaching expenses • License cost (Anaconda/ Cuda, model related) • Checking cost • Prototype cost
Score	20 P	20 P	20 P	20 P	20 P

1. Different tools are deployed in different enterprise systems, and different criteria need to be defined. But all the guidelines should be based on the fine-grained definitions above. Example: (a) Deployment data not transparent → 0P. (b) Transparent but user could determine when a deployment occurred → 3P. (c) Completely transparent. → 5P.
2. The total score should be ideally 100 and arithmetic style is addition.

Figure 2: Our Six Evaluation Metrics. Final score can be calculated from Six Metrics (Learning cost for engineers, Transparency, Stability, Security, Parallelism and Cost) and each standard could be calculated by using the referred points. The standards are derived from Engineer experiences and summaries. The final conclusion could be based on real industry needs and could be weight differently according to real scenarios, as displayed visually in the score distribution graph (e.g. blue line for Openshift and red for Kubernetes& ArgoCD as example). When the score is 100, it will appear as a regular pentagon.

Table 2: Sample comparison between OpenShift and Kubernetes based on Six Evaluation Metrics (10) (9).

| Metrics	OpenShift
Learning cost for engineers	Openshift is a product from IBM running on top of Kubernetes with other CI/CD integration like Jenkins and Ansible (9)
Transparency	Image Streams and Service Catalog make OpenShift better in terms of management, while limited logs also prohibiting advanced transparency (9)
Stability and Security	Often occurs Out of Memory Killed error, very strict in security and need to certain level of permission to maintain minimum security level
Parallelism	Very high level of parallelism, OpenShift provides better support to users, also enable of parallel deployment activities
Cost	Price is different for individual developer or entrepreneur, self-developer is free, but expensive for entrepreneur in terms of resources (9)
Metrics	Kubernetes
Learning cost for engineers	Kubernetes is an open-source container orchestration tool fitting for industry project (10)
Transparency	Management of container images is not so easy in Kubernetes, also Kubernetes service catalog has less provision of services within clusters, however with help of ArgoCD or K9s can easily access to logs of deployment (13)
Stability and Security	Very stable once container or pod is successfully deployed, easy to maintain security level
Parallelism	Quiet limited, often requires additional tools for good deployment experience
Cost	Open-source tool, often not as single solution, needs third part plugins and cloud providers (10)
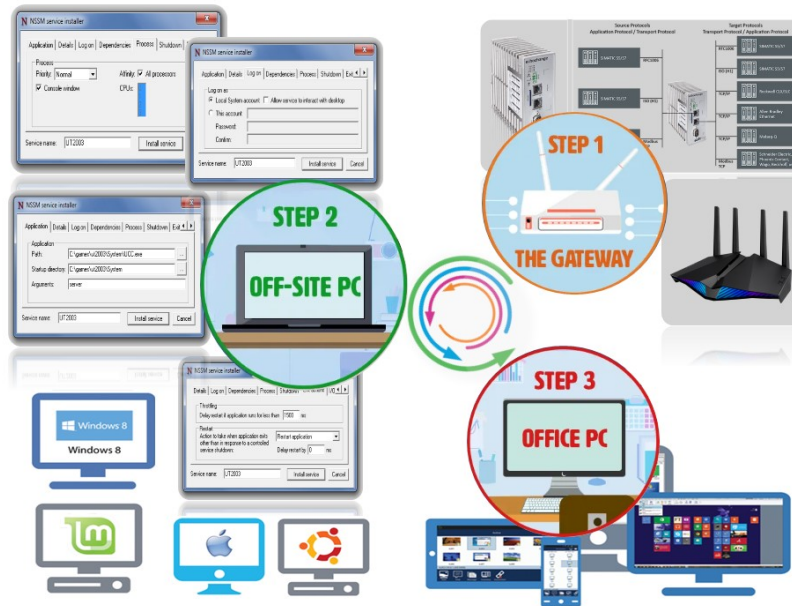
Figure 3: NSSM Batch Service Pipeline. The Net-support Manager and NSSM Batch are usually utilized here for static deployment. The first step needs to purchase the Network Gateway system, such as IoT Gateway or industrial Network Gateway. The gateway usually contains not only the hardware device but also network protocols. then, it is about the set up of the NSSM service to run the inference script once off sit PC is started using local CPU/GPU resources. Moreover, Office PC can finally access off-site PC via Net-support Manager, which supports almost all 20+ systems. AI developers can finish the remote deployment of ML systems even without the Knowledge of real production environments.(7)
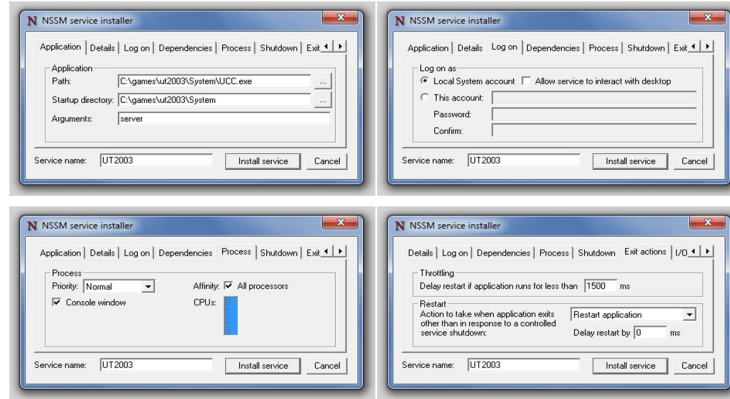
Figure 4: Batch Service configuration (7). These four figures summarize the setup of the Batch Service configuration. The top left illustrates the application path and start-up directory, e.g., it could be a python inference script or executable software path. The top right indicates the service logs and can be placed under the same project folder path. We can set the process-related settings such as CPU resources, as shown in the bottom left, and service restarting times, as captured in the bottom right. It is evident that NSSM services have many steps of configurations set, more steps or parameters as documented in following Figure 5.

Windows Services	REM Preset	Code
Application	REM Application Path	nssm get MyService Application
	REM Working Directory	nssm get MyService AppDirectory
	REM Application Parameters	nssm get MyService AppParameters
Details	Display Name of the REM service	nssm get MyService DisplayName
	REM Service Detailed Description	nssm get MyService Description
	Way to start the REM service (AUTO_START)	nssm get MyService Start
Log on	Way to start the REM service (LocalSystem)	nssm get MyService ObjectName
	Way to start the REM service (WINX_OWN_PROCESS)	nssm get MyService Type
Dependencies	REM Dependent Service	nssm get MyService DependOnService
Process	REM Execution Priority	nssm get MyService AppPriority
	Whether REM should not start a hidden Console window to execute the service	nssm get MyService AppNoConsole
	CPU Affinity	nssm get MyService AppAffinity
Shutdown	Way to send the signal to the application	nssm get MyService AppStopMethodSkip
	Gracefully Shutdown for Control-C (Preset: 1500ms)	nssm get MyService AppStopMethodConsole
	Gracefully Shutdown for WM_CLOSE (Preset: 1500ms)	nssm get MyService AppStopMethodWindow
	Gracefully Shutdown for WM_QUIT (Preset: 1500ms)	nssm get MyService AppStopMethodThreads
Exit actions	REM Delay Startup if program dies within milliseconds	nssm get MyService AppThrottle
	Default Behavior when REM application ends	nssm get MyService AppExit Default
	Delay Start milliseconds for REM service restart	nssm get MyService AppRestartDelay
I/O	The file to which STDOUT messages of the REM application are output	nssm get MyService AppStdout
	The file to which STDERR messages from REM applications are output	nssm get MyService AppStderr
File rotation	Rotate the log file automatically	nssm get MyService AppRotateFiles
	Way to create REM STDOUT message log files	nssm get MyService AppStdoutCreationDisposition
	Way to create REM STDERR message log files	nssm get MyService AppStderrCreationDisposition
Environment	Set the environment variables before starting the REM service	nssm get MyService AppEnvironmentExtra

Figure 5: NSSM Command Window Description (7). Equivalent command examples upper show the commands which would configure an existing service to NSSM Command Window. If the application needs to start in a particular directory you can enter it in the Startup directory field. If the field is left blank the default startup directory will be the directory containing the application. Initial parameters are often default parameters.
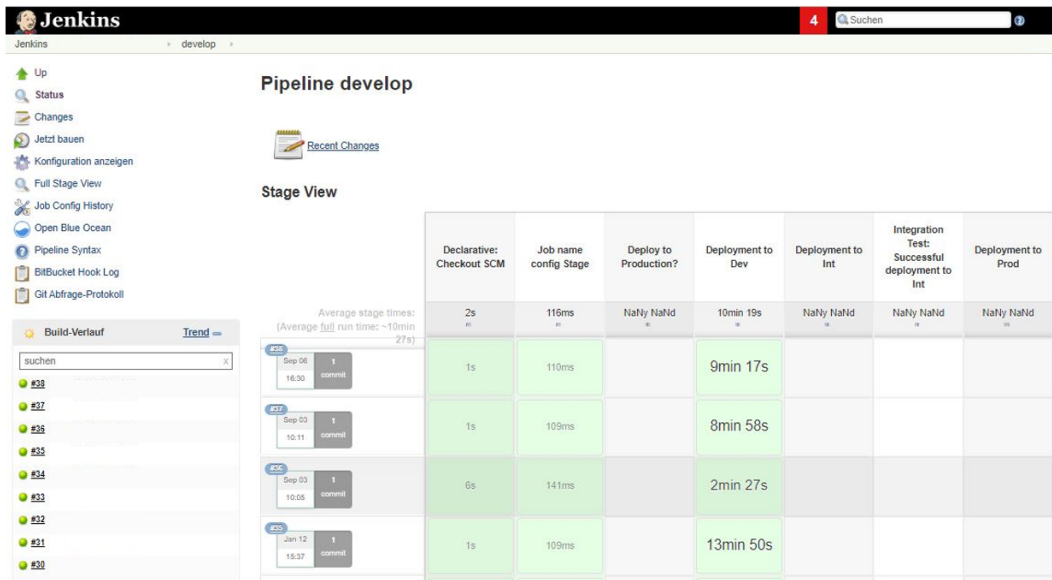
Figure 6: Jenkins Deployment User Interface (Image source from (5)). As an open-source CI automation server, Jenkins can be used to automate all sorts of tasks related to building, testing, and delivering or deploying (5). Jenkins can invoke docker containers on agents/nodes (from a Jenkinsfile or through Blue Ocean) to build AI Pipeline projects, namely, write a Jenkins script (Jenkinsfile is recommended) and clearly define each stage and step, then add credential into the requirment.txt file. As shown in the figure, from Graphic Jenkins Pipeline, we can easily trigger the docker image build and track the stage of current deployment in real-time (5).
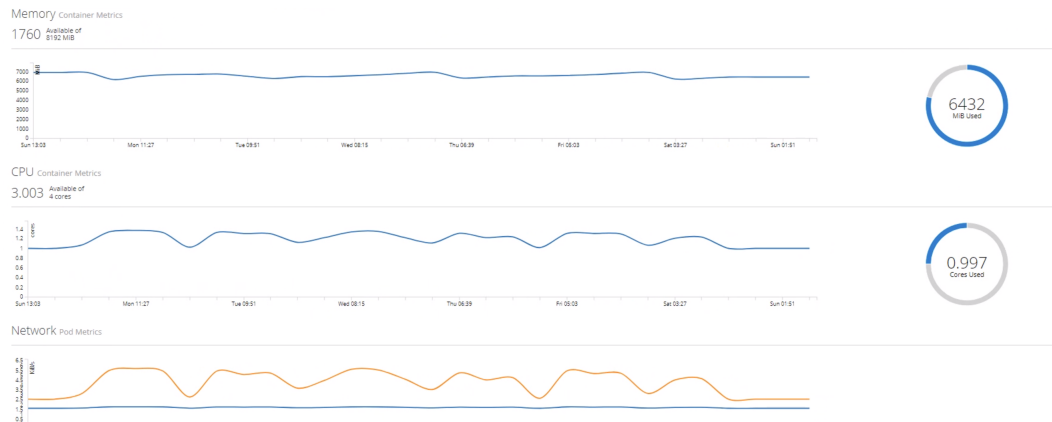

Figure 7: Use case resource consumption logs based on PaaS OpenShift deployment methods (9). These three logs perfectly illustrate the consumption of a typical anomaly detection use case running on Openshift. Use-case deployed through this method often encounters the OOM (Out of Memory Killed) error, while the docker image size is usually up to 3 to 4 GiB. Moreover, the image size is up to 4MB. The setting for 4 to 6 CPU cores limit sometimes is also not enough. From practical experiments and experiences, the PyTorch weight is much less than TensorFlow save model weight. Although the memory composition also depends on the post-processing logic complexity and model amount, frequent OOM Killed issues while saving many useless network layer logs make people less trusted in the stability of the AI systems.
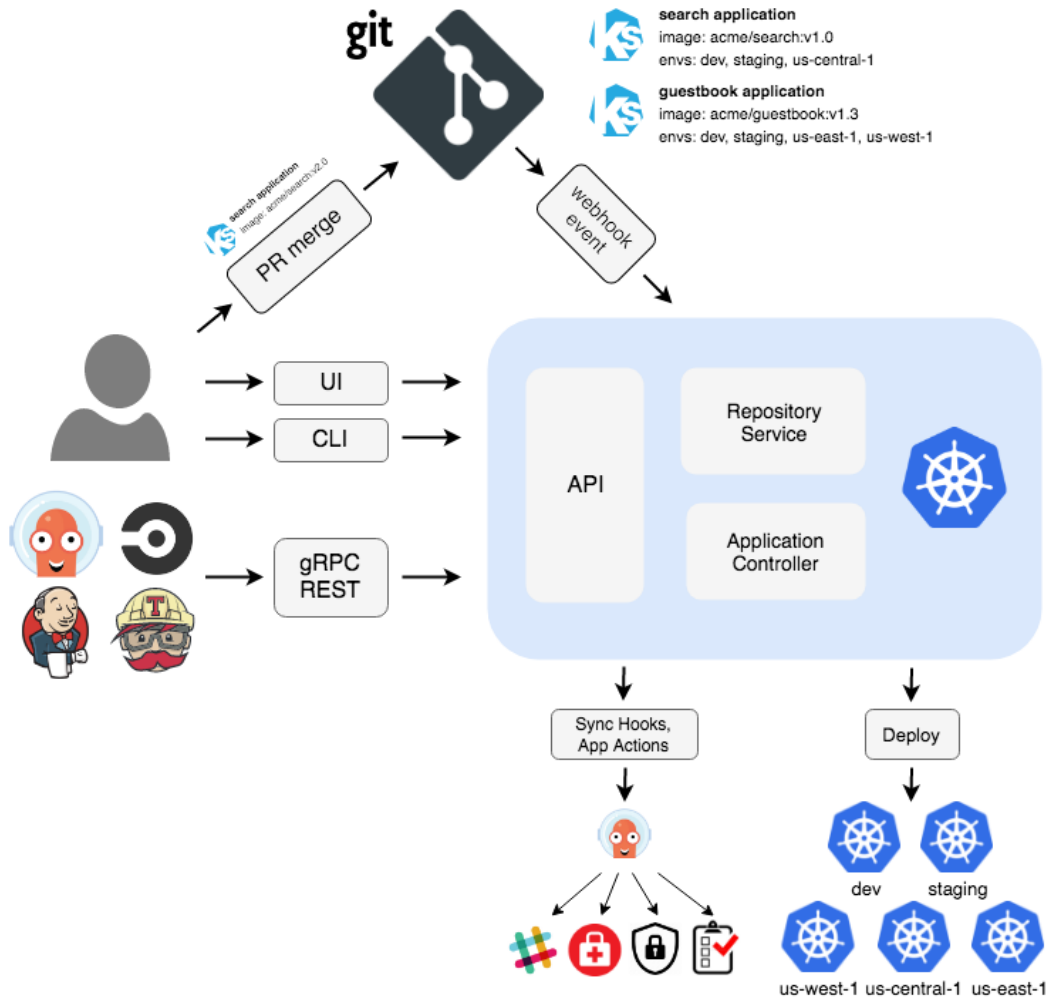
9

Figure 8: ArgoCD CI/CD pipeline (13). The Graph summarizes the structure of the Repository container and Kubernetes-based deployment in great detail. From left to right, tools like ArgoCD and Jenkins continuously monitor all running applications and serve as Kubernetes controllers. When the AI developer builds the docker image and pushes the docker image to the cloud repository, the only thing left is to trigger the pull request. ArgoCD will compare their live state to the desired state specified in the Git repository; once it is differentiated, it will automatically start pulling the latest docker files and all necessary configuration files and start the SYN and Deploy. Here, Kubernetes is responsible for the Deployment and Management of infrastructure. For a detained explanation of docker image, you may refer to following Figure 9.
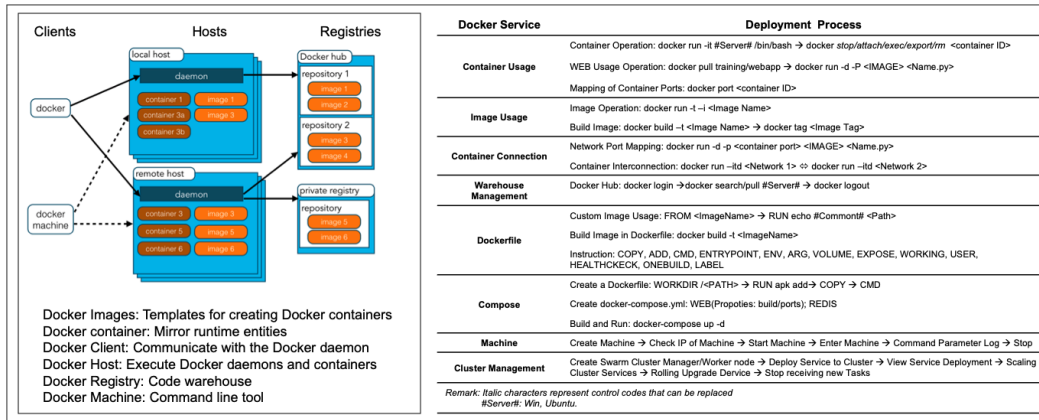
Figure 9: Docker architecture. Docker image is equivalent to a root file system. The relationship between Image and Container is like classes and instances in object-oriented programming (14). Image is a static definition, and a container is an entity when the image is run. Containers can be created, started, stopped, deleted, suspended, etc. Repository can be regarded as a code control center to store images. The rest of the units are well explained in the graph upper.
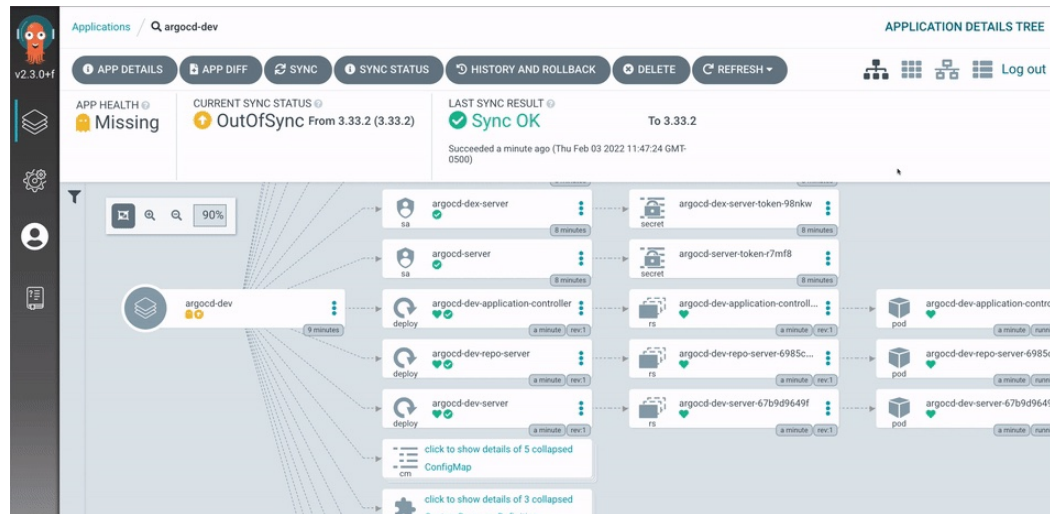

Figure 10: ArgoCD Application User Interface (Image source from (10)). As a declarative, GitOps continuous delivery tool for Kubernetes (10), ArgoCD can track the application status in Kubernetes Repository (13). and automates the deployment of the desired application states to specified target environments. It can update branches, tags, or pins to a specific version of manifests with a Git commit(13). The graph shows that the Web UI provides a real-time view of application activity. It supports multiple config management/templating tools (Customize, Helm, Jsonnet, plain-YAML)(13). While it also requires permission to perform centrin level of delete/refresh/edit/SYN) actions as well.