Automation for Future Conflicts

The requirement for infrastructure and processes to enable the software development lifecycle

by Maj David "Skip" McGee

>Maj McGee, is a Communications Officer currently assigned to the Marine Coders section, Innovation Laboratory Branch in the Marine Innovation Unit. Maj McGee previously deployed as the Officer in Charge of the Deployable Joint Command and Control Detachment 20.2. He has served with MARFORPAC as a C4 Operations Officer, as the Operations Officer for Marine Wing Communications Squadron-48 Det A Fwd, as the S-6 Director for Marine Corps Air Station Yuma, and held Platoon and Company Command at 8th Communication Battalion.

In *Force Design 2030*, Commandant Berger identifies an imperative requirement to modernize the force,[1] "[F]uture Marines will possess ... the intellectual and technical skills required to innovate, adapt, and succeed in the rapidly changing 21st century operating environment."[2] In a technology-dominated operating environment, automation is essential to mission success.[3] The Russian war with Ukraine produced software applications enabling decentralized targeting and automated alerting.[4] Innovation through automation provides a leaner, increasingly efficient, and effective fighting force. The requisite technical infrastructure and software lifecycle process do not currently exist to enable Marines across the force to effectively automate solutions to current and future problems. What infrastructure and processes should be developed to enable the development of automation within the Marine Corps?

Successful Application Development Example

While writing a deployment's worth of fitness reports over a satellite connection from a tent in Jordan, I was frustrated that the connection periodically failed. The first couple of times I was informed that the weather at the distant end was bad so there was not any point in troubleshooting. Marines are accustomed to adapting to marginal conditions. After a while, I started wondering why the weather in Lago, Italy, was so bad. Is this weather problem real a convenient answer for the satellite controllers to avoid messing with power or troubleshooting the connection? It was perpetually dry and sunny in Jordan, so at least half of the satellite shot seemed to be without weather impacts. I checked the weather after the connection was restored and quickly found that the weather issues the controllers reported did not in any way match the weather reporting. This problem became so frequent that I kept browser tabs dedicated to the weather at different locations. Eventually, I was tired of asking my Marines to get past the weather story. I wrote a short Python application to concurrently display the weather at two locations using data from a free weather Application Programming Interface (API). I turned this application into a Windows executable, and one of my Marines wrote a PowerShell script to sign the application with the domain certificates for use on our laptops.[5] Shared network storage enabled distribution of the application to any user who desired to run the executable. This application development, testing, and delivery lifecycle worked because the infrastructure (domain, servers, and workstations) was entirely maintained and administered by my unit. We possessed both the requisite infrastructure and the capability to develop a software lifecycle process. While five detachment rotations later this application may no longer be used, this experience demonstrates that small problems can be solved or reduced through automation.

External Federal Application Development Example

Looking outside the Marine Corps, here is an example of an existing software development lifecycle in a different federal agency. Developers who work on an application test and change their code locally and then commit their changes into a GitLab repository. As code is added/changed and committed to the code repository, the repository's continuous integration pipeline uses runners to build the code into a Red Hat Package Manager package and run applicable functional tests to ensure that the changes do not negatively affect the application's performance. The pipeline continues to execute other jobs, such as checking for dependency vulnerabilities and security concerns with Static Application Security Testing tools and potentially with dynamic testing tools. Assuming that problems are not detected, the continuous delivery portion of the pipeline then signs the Red Hat Package Manager and

deploys the package to a development environment yum server repository. When the hosts in that environment run a periodic update, they update to the latest application version. This process occurs seamlessly, without manual intervention, unless there is a need for developer attention to resolve a functional issue or security problem. Additional functional and dynamic tools are run against the new version of the application in the development environment up to and potentially even including user testing. At some point in this review process, another pipeline is triggered to push the application from the development environment to the production environment repository where the production hosts update to the new version. Is there a reason that we could not create a similar infrastructure and software development process within the Marine Corps Enterprise Network (MCEN)?

Failed Application Development Example

The MARFORPAC G-6 watch described their process for obtaining intelligence and awareness of cyber threats and threat actors in the INDOPACOM Area Of Responsibility. Commercially procured and tailored threat intel data was too expensive, so they maintained a list of over thirty uniform resource indicators or websites that they read daily in order to identify changes and relevant events in the area of responsibility. This task was time intensive, depended entirely on the analyst for its thoroughness, and did not scale well as websites or uniform resource indicators were added. I saw an opportunity for automation and wrote a Python web scraper to identify occurrences of the keywords that they were looking for within the website list and return the corresponding paragraphs, so the analysts only needed to review a relevant subset of websites instead of iterating over the whole list. This solution could enable the watch to increase its list of uniform resource indicators and improve its area of responsibility awareness while both reducing the time spent browsing the list and standardizing the review across watch officers. Yet, I could not find a path to integrate this script into the MCEN for the watch officers to use. Lack of hosting resources, proxy problems, and lack of shared authentication were the challenges. Since I could not identify the approval process or automate the distribution of a software product into the MCEN, the initial code was migrated from DevForce to GitHub.[6] However, this meant that all work had to be conducted outside of the MCEN due to the inability to access GitHub to make code changes and work on the app from inside the MCEN. Once the first conceptual version of the application was ready for testing, I could not find an effective solution for hosting it or identify the process for hosting such an application in the MCEN. I considered using a raspberry pi to host a web user interface based on the script on my home network but decided that the lifecycle maintenance of the application and user accounts would be too much to support on my own for free. This project failed due to the lack of hosting infrastructure and process for development, testing, approval, and delivery.

Is there a reason that we could not create a similar infrastructure and software development process within the Marine Corps Enterprise Network (MCEN)?

Sharing Solutions

Building infrastructure to enable automation is a shared joint problem. The other Services understand the 21st-century operating environment and the value of automation. The MCEN now makes VSCode, Anaconda, and RGui available to end workstation users in the software center—so the suite of user development tools is growing. The Defense Information Systems Agency's GitLab instance is a significant step toward a SecDevOps infrastructure that enables joint application development. This resource is accessible from inside the DOD Information Networks, freely available to users who desire to host a project or repository, and uses common access card authentication as well as personal access tokens for pushing/pulling code changes. GitLab runners (if unfamiliar, think computation and processing for continuous integration jobs) can be registered to this instance to enable building software from the code repositories using continuous integration/continuous delivery pipelines, enabling a developer or developer team to build an application entirely inside the DOD Information Networks.

From a knowledge and capability perspective, integration with the Reserve Component can provide expertise using the existing initiatives of the Marine Corps Software Factory and the Marine Coders.[7] The 06XX community possesses the 0673 MOS, which is developing the pipeline to train Marines.[8] Simultaneously, the coding and automation skills of the average Marine are advancing as programming courses increase in popularity in high schools and colleges.[9] Project this trend into the next ten to fifteen years and the ability of a Marine to automate a problem will be correspondingly higher. We must develop the infrastructure and processes to weaponize that ability.

Analyzing the Problem

While a case could be made for Service-specific GitLab/GitHub instance, we will assume here that the DISA infrastructure remains freely available to any service member. The remaining challenge, therefore, is integrating the continuous delivery portion of a pipeline into the MCEN. This process begins by registering runners inside the MCEN. Then we need to answer some organizational questions to determine the way forward, such as how do we authorize and deploy applications? What are the resource requirements and what is the secure delivery process? Web applications, applications signed

with DOD Certification Authority certificates, and corresponding web access firewall implementations could add a requirement for closer coordination and support from system administrators and potentially manual intervention into the deployment cycle. How does that application development cycle occur quickly and securely?

There are essentially two potential destinations for these applications, a test/development environment and a production environment. The infrastructure for these two potential destinations does not exist (to my knowledge) but would be relatively easy to create, potentially via defined network rules with access to create FedRAMP-approved Red Hat Enterprise Linux virtual machines in Azure or Amazon Web Services to use as runners and application hosts (with some additional security controls around the production environment).

There are three different distribution methods that should be considered, broken down by operating system package manager: a yum/dnf repository for Red Hat Enterprise Linux hosts (assuming an Aptitude repository is unnecessary), integration into System Center Configuration Manager for Windows, and container repository integration. For the moment, we can ignore the container distribution method.The knowledge base or infrastructure of containers, container runtimes, and container registries is currently not resident within the MCEN and FMF, and creating both that knowledge base and infrastructure is a much heavier lift than the solutions I propose.

There are two different application use cases that need to be considered, distinguishing whether that application is deployed to a server or a workstation. Server applications could be hosted in an environment with authentication, defined/limited network access rules, and Domain Name System integration, whereas workstation applications would need to be installed and tested on workstations, presumably requiring local administrative access to the workstation.

While each of these components of the application development cycle possesses unique characteristics and technical problem subsets, the overarching problems that the Marine Corps must solve are infrastructure ownership, defining the application lifecycle process, and funding the supporting infrastructure. The overall resources required to maintain the infrastructure described here are very minimal, not more than one full-time equivalent employee or military member, and some associated costs for virtual machine licensing in Amazon Web Services or Azure. The most critical problems are determining the ownership and application lifecycle process.

The Marine Corps must ... enable software development, testing, integration, and delivery.

Conclusion

Consider a logistician who develops an application or script to help automate a transportation problem and several users in combat operation centers worldwide want to install and use it. How would that Marine accomplish that task right now? Would they contact people at Information, Command, Control, Communications and Computers and Marine Corps Cyberspace Operations Group and try to work their way uphill to develop a development process and infrastructure? Quite possibly they would give up in frustration once someone mentions the most dreaded three letters in military information technology: the ATO (authority to operate). How should they tackle that problem? We need a defined process and infrastructure for completing that software development lifecycle at the pace and timeline of the warfighter. Commandant Berger ordered us to innovate, adapt, and succeed. The Marine Corps could lead the Services in developing secure coding practices and secure application delivery practices and processes because we have the capability to do better. We are growing the requisite knowledge base across the force. We do not currently possess the Service-level infrastructure to enable secure automation and application development. The Marine Corps must develop a resourced testing/development environment and define the approval process to enable software development, testing, integration, and delivery.

Notes

1. David H Berger, *Force Design 2030* (Washington, DC: May 2020).

2. Ibid.

3. Charlie S. Bahk, "Announcement of the Marine Corps Software Factory Pilot," *Marines.mil*, March 30, 2023, https://www.marines.mil/News/Messages/Messages-Display/Article/3325426/announcement-of-the-marine-corps-software-factory-pilot.

4. Drew Harwell, "Instead of Consumer Software, Ukraine's Tech Workers Build Apps of War," *Washington Post*, March 24, 2022, https://www.washingtonpost.com/technology/2022/03/24/ukraine-war-apps-russian-invasion.

5. David McGee, "WeatherScraper," *GitHub*, November 29, 2020, https://github.com/skipmcgee/weatherscraper.

6. This resource https://gitlab.devforce.disa.mil was last accessed in July of 2022 and appears to no longer be accessible, perhaps replaced by https://web.git.mil.

7. "Announcement of the Marine Corps Software Factory Pilot."

8. Ibid.

9. Additional information is available at https://advocacy.code.org/2022_state_of_cs.pdf.