REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	**2. REPORT TYPE**	**3. DATES COVERED** *(From - To)*
07/01/2019	Technical Report	

4. TITLE AND SUBTITLE
Kessel Run Release Engineering and Continuous Delivery Pipelines

5a. CONTRACT NUMBER
FA8702-19-C-0001

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Reynolds, Benjamin A.

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
The MITRE Corporation
202 Burlington Road
Bedford, MA 01730-1420

8. PERFORMING ORGANIZATION REPORT NUMBER
PRS-19-1582

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Air Force Materiel Command

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
DISTRIBUTION STATEMENT A. Approved for public release: distribution unlimited.

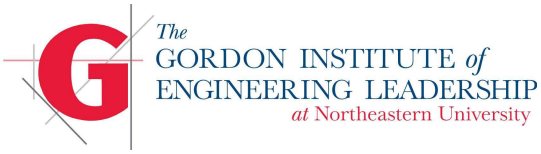
13. SUPPLEMENTARY NOTES
A candidate challenge project for The Gordon Institute of Engineering Leadership in progress towards the requirements for the Graduate Graduate Certificate in Engineering Leadership Northeastern University, Boston, Massachusetts.

14. ABSTRACT
This project developed standards and automation for release engineering at Kessel Run. An automated release pipeline allows application development teams to build, test, package, and release their applications in a consistent and reliable manner with minimal effort. A corresponding deployment pipeline can then be run by platform operators to automatically deploy applications into production using the artifact produced by the release pipeline. These pipelines, written in Python and for use on the Concourse automation tool, eliminate tremendous amounts of manual work, saving time and reducing the possibility of human error.

15. SUBJECT TERMS
Software Engineering (General); standards; Kessel Run; automation;

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			Susan Carpenito
				59	**19b. TELEPHONE NUMBER** *(Include area code)* 781-271-7646

Kessel Run Release Engineering & Continuous Delivery Pipelines



A Challenge Project Final Report presented

by

Benjamin Reynolds

of

MITRE

to

The Gordon Institute of Engineering Leadership

in progress towards the requirements

for

the Graduate Certificate in Engineering Leadership

Northeastern University

Boston, Massachusetts

July 2019

DISCLAIMER

The views expressed are those of the author and do not necessarily reflect the official policy or position of the Air Force, the Department of Defense or the U.S. Government.

NOTICE

1 ABSTRACT

The Kessel Run program is a software startup within the Air Force with a goal of continuously delivering valuable software that airmen love. One area in which the Air Force looked to MITRE for support accomplishing this mission is release engineering.

"Release engineering is a relatively new and fast-growing discipline of software engineering that can be concisely described as building and delivering software. Release engineers … define all the steps required to release software—from how the software is stored in the source code repository, to build rules for compilation, to how testing, packaging, and deployment are conducted."[1]

Throughout this project, the Gordon Candidate, Benjamin Reynolds, developed standards and automation for release engineering at Kessel Run. An automated release pipeline allows application development teams to build, test, package, and release their applications in a consistent and reliable manner with minimal effort. A corresponding deployment pipeline can then be run by platform operators to automatically deploy applications into production using the artifact produced by the release pipeline. These pipelines, written in Python and for use on the Concourse automation tool, eliminate tremendous amounts of manual work, saving time and reducing the possibility of human error.

Legacy air operations software can take months to deploy into production. Through the introduction of platform as a service and cloud native applications, Kessel Run reduced that time to roughly 2-3 days. As a result of this project, the time to onboard applications to a production site has been reduced even farther to approximately 2-3 hours. Repeat deployments of updated applications has been reduced from approximately 2 hours to 30 minutes. Other benefits as a result of the project include the ability to test new instantiations of the platform using automated application deployment, shared knowledge across teams due to a standard language and process, repeatability and auditability through automation, and the breaking down of communication barriers to promote the first DevOps culture in the Air Force.

Furthermore, this project provided many leadership growth opportunities for the Gordon Candidate. Benjamin practiced communication and advocacy to build trust and buy-in with a wide array of stakeholders in support of this project. He then had to employ and grow in nearly all of the other leadership capabilities as he worked to realize the vision which he developed for what would make this project a success.

[1] (Beyer, et al. 2016)

2 AUTHOR'S BIO

Benjamin Reynolds is a Software Systems Engineer and Gordon Fellow Candidate pursuing a Master of Science in Computer Systems Engineering. As an undergraduate student at Rensselaer Polytechnic Institute, Ben received a dual Bachelor of Science Degree in Computer Systems Engineering and Computer Science. During his time at RPI, Ben also completed three internships in his field at three different companies. He lives his life in accordance with a quote by Abraham Lincoln which states, "Whatever you are, be a good one." Ben always gives his best to the task at hand and believes that if something is worth doing, it is worth doing well.

Ben first received experience in Continuous Integration and Delivery during a summer internship with Workday Inc. where he worked with two other interns to develop an automated testing platform controllable through the Slack chat application. This project enabled software developers at Workday to run test suites and perform integration tasks with a simple message to a bot in Slack which would automatically respond with results. Though he didn't know it at the time, this introduction to agile DevOps and automated testing laid the foundation for what would become a full-time role at MITRE.

Ben started as a full-time employee at MITRE right as the Air Operations Center Weapon System (AOC-WS) 10.2 program was being cancelled and a new Pathfinder Initiative (Kessel Run) was beginning to take its place. Recognizing the shift in expertise which would be needed to remain effective in their support of the program, MITRE brought Ben on board to become their expert in Continuous Integration and Continuous Delivery (CI/CD) concepts (key components in the agile approach taken by Kessel Run). In his first months with MITRE and on the AOC program, Ben spent the bulk of his time learning all he could about these concepts and modern software development in general. Through multiple prototyping efforts and over 30 presentations to Air Force leadership, he earned a role as a Subject Matter Expert (SME) in Continuous Integration and Delivery and software automation before beginning to work directly with the Air Force sponsor to automate the deployment of their cloud environment. This role on the Kessel Run platform team gave Ben the opportunity to lead a diverse group of engineers in achieving a complex technical goal of complete platform automation.

Beyond the technical expertise which Ben has acquired through internships and his experience at MITRE thus far, he also has leadership experience which enables him to bring projects to success. In high-school, Ben served two summers as a teen leadership trainee at a summer camp before working as a full-counselor the following year. He gained valuable training in motivating people and building and maintaining relationships. He also took an honors class in his senior year where he organized and led community events including a Spook Trail fundraiser for a scholarship fund, a voter registration drive, and a Special Olympics event for nearly 20 other schools in the area. In each of these cases, Ben functioned as leader of a team to organize and operate a complex event.

With a strong background of both technical and leadership experience, Benjamin Reynolds possessed the appropriate foundation to bring this challenge project to success.

His drive for excellence and prior experience provided him the ability to overcome challenges and ensure that this project would be completed on time and provide value to the sponsor organization.

3 ACKNOWLEDGEMENTS

This work would not have been possible without the financial support and generosity of MITRE in funding my graduate education. I would like to express my deepest appreciation to my Industry Sponsor Advocate, who helped me to receive this financial backing, served as my industry sponsor advocate for the challenge project, and convinced me to join the Gordon program in the first place. Your support in both work and school was instrumental in the success of this project.

I also wish to extend sincere thanks to my Gordon mentor and faculty advisor, as well as the rest of the Gordon faculty who kept me on track and guided the direction and quality of this project from the very beginning. The time you sacrificed and advice you offered are evident in the outcome of the project.

I'm grateful to all of the individuals within Kessel Run who have given time to support meetings, provide feedback, and work with me to implement this project across the organization. Without your willingness to take a chance on this project and integrate it into your day to day work, the project would be nothing.

To my fellow Gordon Candidates, thank you for the teamwork, great memories, and lasting friendships which have been built as we all worked together to succeed in this program.

Lastly, nobody has been more important to me in pursuing this project than my wife. Thank you for your patience, encouragement, and love as I endured the long nights, busy weekends, and stressful weeks. This tremendous achievement would not have been possible without you.

4 TABLE OF CONTENTS

5 LIST OF FIGURES AND TABLES

6 INTRODUCTION

The Kessel Run program is revolutionizing the way the Air Force builds and delivers software. In traditional Department of Defense (DoD) acquisitions programs, large contracts are put out for bids among contractors with extensive and detailed lists of requirements. The winning contractor then develops the system, puts it through test, and delivers it back to the government. In the world of software, this approach is referred to as waterfall and is generally considered outdated. The waterfall approach rarely delivers software that truly meets the end users' needs and often ends up running way over on both schedule and budget. Kessel Run is taking a different approach to software delivery, known as agile. Agile emphasizes minimal requirements, direct input from real users, and fast feedback loops, ensuring that the final product is both useful and well-liked by the users. In order to achieve this frequent feedback and fast iteration, agile software organizations make extensive use of automation to build, test, deliver, and deploy applications.

The engineers responsible for these practices and the automation surrounding them are often referred to as release engineers. "Release engineering is a relatively new and fast-growing discipline of software engineering that can be concisely described as building and delivering software. Release engineers have a solid (if not expert) understanding of source code management, compilers, build configuration languages, automated build tools, package managers, and installers. Their skill set includes deep knowledge of multiple domains: development, configuration management, test integration, system administration, and customer support… Release engineering is a specific job function at Google. Release engineers work with software engineers (SWEs) in product development and [site reliability engineers] SREs to define all the steps required to release software—from how the software is stored in the source code repository, to build rules for compilation, to how testing, packaging, and deployment are conducted."[2]

Prior to this project, Kessel Run had no rigor around release engineering. Each application team had a different approach to the way they released their applications, posing significant challenges within the organization. No standard storage location for releases meant that platform operators were constantly searching through several different places for the artifacts they needed to deploy. A lack of standard versioning practice meant that there was no way to tell which release of an application was the newer one or which version was currently running in production. Lack of automation for deployments led to operators spending hours communicating with application teams and performing manual configurations and deployments. Lack of common practice for compiling applications made it nearly impossible for a newcomer to the organization to download and build an application from source without extensive guidance. These are just a few of the many problems which prompted the need for a standard to release engineering across Kessel Run, ultimately becoming the driving force for this project.

[2] (Beyer, et al. 2016)

The initial scope of this project was to develop standards, along with automated release and deployment pipelines which would eliminate many of the challenges Kessel Run faced when releasing applications. The high-level requirements for a minimum viable product (MVP) at inception were as follows:

- Must work for the ~20 application teams in the organization
- Must require minimal engineering effort from application teams to adopt the new process
- Must be automated to the extent possible
- Must support automated application deployment on a classified network
- Must standardize the way teams build, version, document, release, and deploy their applications

The technical approach taken to meet these requirements was to develop two separate, automated pipelines using Concourse. Concourse is an open source CI/CD automation tool which was already being used as the primary CI/CD orchestrator on Kessel Run. Since it is developed by Pivotal, Concourse also provides seamless integration with Pivotal Cloud Foundry (PCF), which is the production platform used by Kessel Run. It is able to support the automation of nearly any workload through the use of Docker images. Each step within the pipeline specifies a Docker image in which the step will run, allowing the developer to identify which dependencies or packages are needed to execute the automation. This type of flexibility lets the developer use whatever scripting languages, command line interfaces (CLIs), or other tools are necessary to best accomplish a given task. For this project, Python was chosen as the scripting language because it can be written very quickly allowing for rapid progress, has a strong community of support, and can be learned easily by anyone who wishes to contribute to the project in the future.

The first of the two pipelines, the release pipeline, operates on the unclassified development network. This pipeline compiles an application from source code, configures the target environment, deploys the app to an unclassified platform for verification, stores the release in an artifact repository, tags the source code as a release, increments version numbers, bundles all the supporting configuration files, documentation, and artifacts, and uploads them as a single tar.gz file to a centralized Simple Storage Solution (S3) bucket in Amazon Web Services (AWS). When this completes successfully, it also notifies platform operators that a new release is available and ready to be deployed to production. Once the tar.gz files are transferred to the Air Force network and put in an Elastic Cloud Store (ECS) bucket by platform operators, the second pipeline begins. The second pipeline is the deployment pipeline which retrieves the tar.gz from ECS, unpacks it, verifies checksums are correct, configures middleware services on the platform, and configures/deploys the application.

This automated approach to application release and deployment has greatly improved the speed at which new capabilities can be fielded to the warfighter. Applications can be freshly deployed to a new platform in hours instead of months as was the case in legacy 10.1 (see section 6.3) development and application upgrades can be deployed in minutes

instead of weeks. This allows developers to iterate faster on valuable solutions, frees up platform operators to focus on platform maintenance and upgrades, and prepares Kessel Run for future growth in the number of application teams and production sites. Overall, this project has allowed Kessel Run to better accomplish its mission of continuously delivering valuable software that airmen love.

Achieving this level of success with the project required use of nearly all the leadership capabilities learned through the Gordon program. The first was vision. To make this project as impactful as it is, the Gordon candidate, Ben Reynolds, had to develop a concrete vision for what the project would accomplish. This vision had to be big enough to address the large and complex problem space yet focused enough to be actionable and achievable. Secondly, establishing support for the project required communication and advocacy. Because this project interfaces with nearly every team in the Kessel Run organization, careful tact had to be taken in communicating the project concept with various stakeholders. The vision had to be communicated to each team in a way that helped them see the direct benefits they would receive. Without this advocacy, the same exact solution may never have made it off the ground. It was only by getting stakeholders to see the value provided by this solution that it took off and reached such success. (The MITRE Corporation 2013)

6.1 Product Mission Statement

Mission Statement: Kessel Run Release Engineering & Continuous Delivery Pipelines	
Product Description	● Standardized processes and automated pipelines for the consistent and scalable release, delivery, and deployment of software applications
Benefit Proposition	● Reduced operator overhead in deploying applications ● Reduced complexity for individual development teams ● Standardized practice across organization ● Automated enforcing of organizational policies ● Release & deployment frequency data publishing
Key Business Goals	● Increase speed & frequency of Kessel Run software development & deployment ● Support scalability and program growth through standardization and automation ● Improve confidence in the delivery of Kessel Run developed software
Primary Market	● Air Force Air Operations Center modernization program
Secondary Markets	● Additional Air Force programs utilizing Kessel Run services
Assumptions & Constraints	● Automation built for automated deployment must be capable of operating on classified network ● Concourse must be used as the primary tool for automation ● Pivotal Cloud Foundry will be the sole deployment platform
Stakeholders	● Application Developers ● Platform / App Operators ● C-Suite Leadership ● Security Team ● Platform Engineering Team

6.2 Project Definitions

AOC-WS	Air Operations Center – Weapon System, command and control center for the Air Force
CI/CD	Continuous Integration / Continuous Delivery, the use of automation to integrate code changes into the code base and deliver them to production
SME	Subject Matter Expert – Individual with recognized expertise in a field or domain
DoD	Department of Defense – United State Government defense agency
MVP	Minimum Viable Product, the simplest implementation which provides an initial value add to the user
Concourse	A CI/CD automation tool developed by Pivotal Software
PCF	Pivotal Cloud Foundry, a Pivotal developed commercial offering of the Cloud Foundry platform
Docker Images	A file which comprises all the layers necessary to run an application or process in a container
CLI	Command Line Interface, a tool used without a graphical interface in the terminal
AWS	Amazon Web Services, the world's largest cloud infrastructure provider
ECS	Elastic Cloud Storage, a Dell product for on-premises blob storage with an S3 compatible API
S3	Simple Storage Solution, an AWS product which provides "infinite" blob storage on the AWS cloud
FFRDC	Federally Funded Research and Development Center, trusted advisor to the government with a charter to provide objective advice and expertise
CATO	Continuous Authority to Operate, security accreditation granted to Kessel Run applications assuming they follow prescribed processes and meet minimum security constraints

Table 1 - Acronyms and Definitions

6.3 Company/Industry Background

6.3.1 MITRE Background

MITRE was originally chartered in 1958 as a not-for-profit company and had a focus on the continental air defense project known as the Semi-Automated Ground Environment (SAGE). This project used some of the earliest computers to connect radar stations, weapon systems, and decision makers in near real time, allowing for better detection of incoming projectiles during the Cold War. SAGE reached operational success in 1963 and led to numerous inventions in computing, software, information displays, and many other areas. Since then, MITRE has grown and now serves a variety of government agencies at the highest levels by managing seven FFRDCs (Federally Funded Research and Development Centers).

The role of an FFRDC is to work in the public interest and serve as a trusted advisor to the government. This can manifest in a variety of ways but often includes providing technical guidance and recommendations, assisting in cost analysis or contracting, and developing prototype solutions to complex problems. For this reason, FFRDCs do not develop products and take them to market for profit as is the case with a typical government contractor such as Raytheon or Lockheed Martin. Instead, FFRDCs perform research and analysis work or prototype development which is provided to the government to address specific needs.[3] There are currently 42 recognized FFRDCs, seven of which are operated under The MITRE Corporation.

MITRE operates using a matrix organization. Within the engineering branch of the company, there are five major organizations. Four of these organizations are project or program oriented and one contains technology centers which are functionally oriented and provide technical expertise to the program centers on an as needed basis. This organizational structure, which is shown in Figure 1 below, allows MITRE to foster both domain knowledge of the government programs through the program-oriented groups as well as deep technical knowledge through the tech centers.

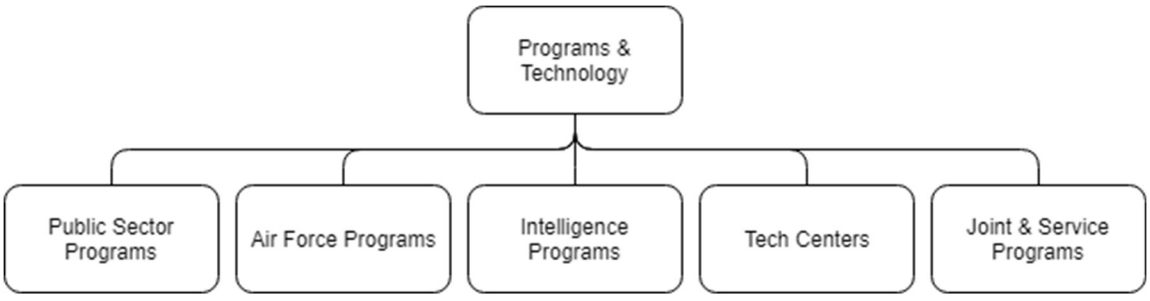


Figure 1 - MITRE Engineering Organizational Structure

[3] ("Our History" 2013)

6.3.2 Kessel Run Background

Much of MITRE's work is done for the Air Force, including the Air Operations Center Weapon System project, or Air Operations Center (AOC) for short. An AOC, of which there are over a dozen world-wide, can be thought of as a command and control center for the Air Force. Each AOC contains over 40 unique, legacy software applications known as the 10.1 baseline, each of which serves a role in planning and executing missions. Figure 2 below shows a typical AOC, this one located at Al Udeid Air Base in Qatar.

Figure 2 - AOC at Al Udeid AFB Qatar[4]

In July of 2017, a significant contract to modernize the software in the AOC (known as the 10.2 program) was cancelled without having successfully delivered a single working application into production. The goal of 10.2 was to modernize the existing 10.1 baseline and ultimately automate the AOC decision cycle depicted in Figure 3 below. The decision cycle, or Air Tasking Cycle as it is officially known, represents the continuous process which the Air Force executes to provide air combat capability.

[4] (United States Air Force 2017)

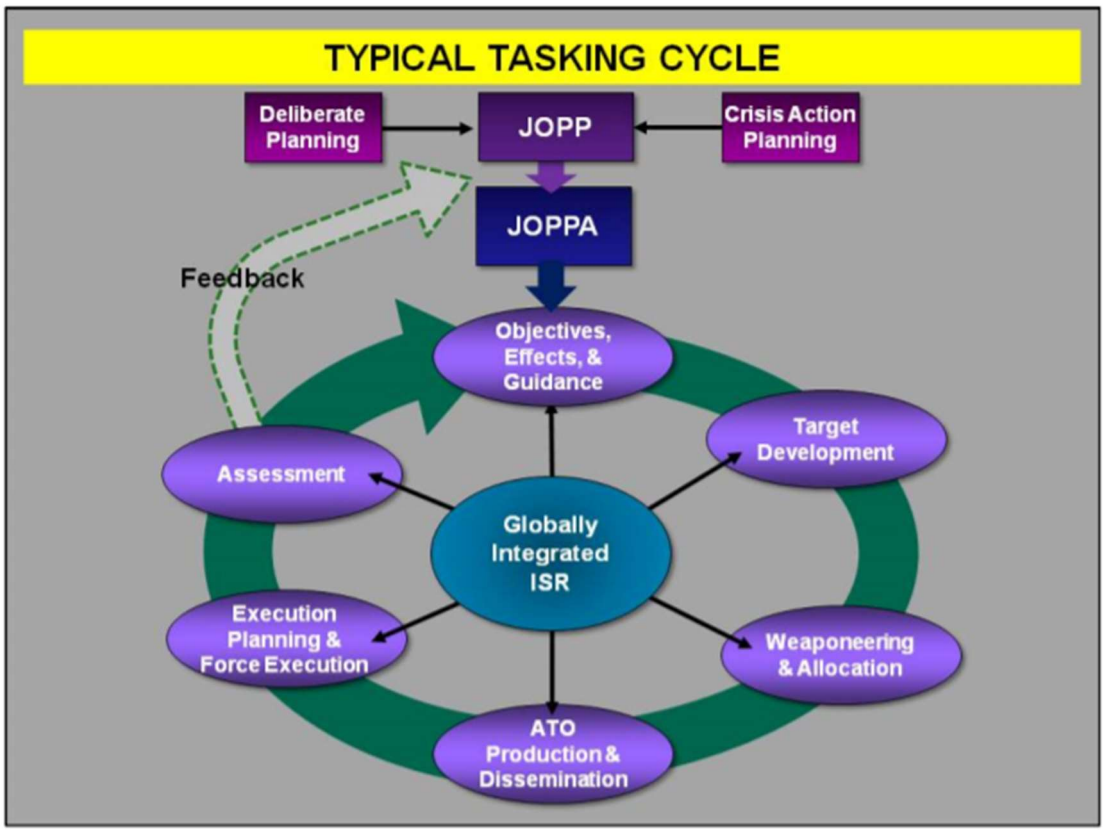


Figure 3 - Air Tasking Cycle[5]

At the time of cancellation, the estimated development costs for 10.2 had risen from an original $374 million to $745 million and the program was already three years behind schedule.[6] This is due largely in part to the waterfall acquisitions strategy traditionally used by the DoD. In this waterfall process, a complex set of requirements is produced and handed off to a contractor to deliver the product. These requirements often left a contractor backed into a corner where there was no room for flexibility and almost no way to deliver on the requirements provided. The process was costly, time consuming, and produced outdated software, often behind schedule. In the case of 10.2, even if the program were to have continued, the software produced as a result would have already become antiquated since the original requirements were written. In response to this, the Air Force started a new modernization initiative known as "Pathfinder" (now Kessel Run). This program would take an agile approach to software development and was intended as an exploratory initiative to see what could be accomplished if many of the traditional DoD processes and red tape were removed. Figure 4 below, although too small to read in detail, depicts the complexity and bureaucratic overhead involved in traditional DoD acquisition which played in role in the failure of 10.2.

[5] (Curtis E Lemay Center For Doctrine Development and Education 2016)
[6] (Baram 2018)

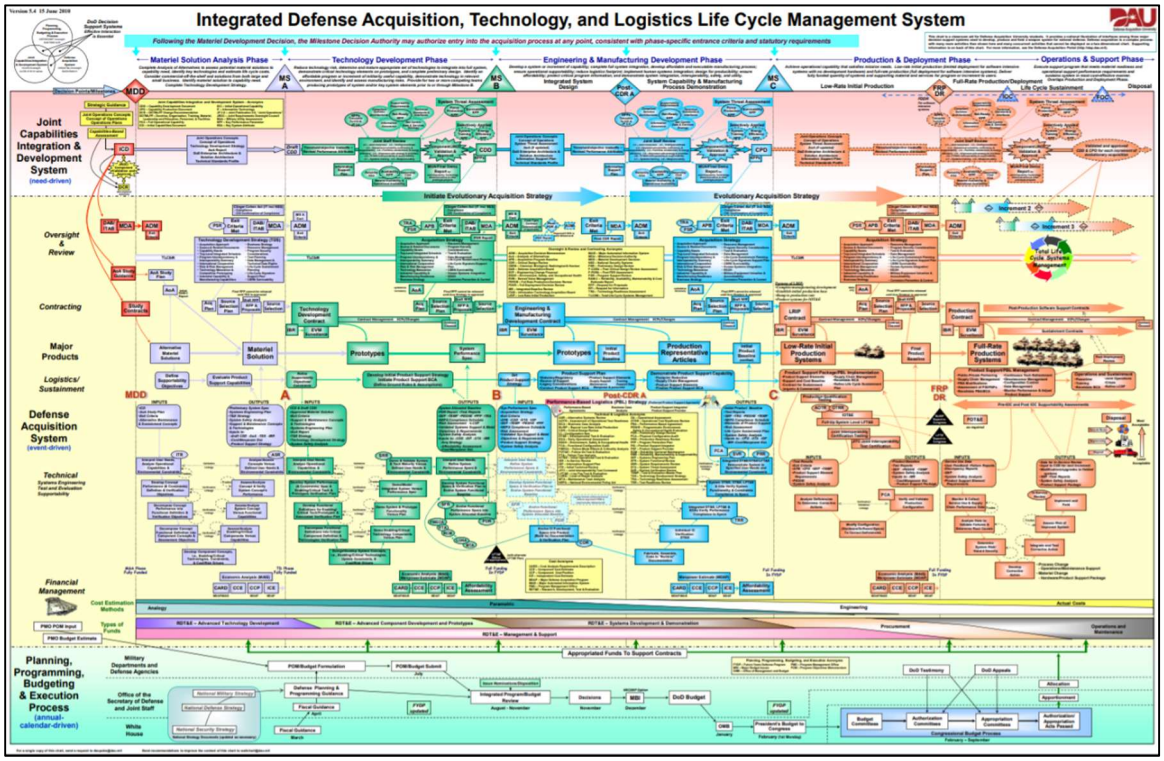


Figure 4 - DoD Acquisition Process Flowchart[7]

Kessel Run is staffed by a blend of government civilians, enlisted Airmen & Officers, contractors, and MITRE employees primarily working collocated at the Kessel Run Experimentation Lab (KREL) in Boston, MA. This collocation provides incredible value over the old method in which the developing contractor was remotely located instead of working hand in hand with the government. The KREL in Boston is the primary development location for Kessel Run and also serves as a major catalyst for the culture shift the Air Force is trying to promote. At the KREL, enlisted airmen and officers wear civilian clothing, work side by side with contractors, and enjoy developing software. The casual, fun culture helps drive the creativity and teamwork needed to successfully deliver high-value software.

Within Kessel Run, work is broken down into three main branches. The Air Ops branch contains the development teams building AOC applications. Wing Ops contains development teams building applications for the maintenance of the F35 fighter jet. Kessel Run Enterprise Services contains all supporting technical services that serve the application teams in the other branches. The Enterprise Services group is further broken down into functional areas focused on specific services and products. Under KR cloud

[7] (Potter 2009)

services, release services is the team where this project falls. Release services is tasked with providing automated solutions for the release, delivery, and deployment of secure applications. Figure 5 below shows this organizational structure.

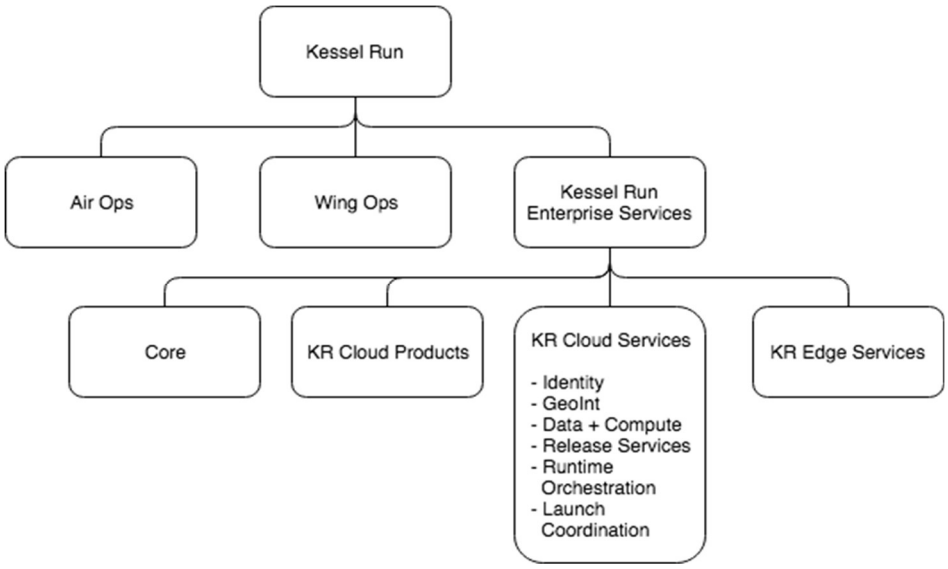


Figure 5 - Kessel Run Organizational Structure

7 MARKET AND IMPACT ASSESSMENT

Speed of delivery has historically been a tremendous problem for government software programs due to security requirements, the DoD acquisition process, and general bureaucracy that slows progress. Though modern software companies have this mostly figured out and can release updates automatically and at will, the government has not yet adopted such an approach. Consider a company like Facebook. When Facebook makes a change to their platform, users phones receive the updated version automatically without any action required on their part. Using continuous integration and continuous delivery (CI/CD) practices, Facebook can push release candidates to mobile users every day.[8] Facebook is then able to receive user feedback on that release and adjust quickly if feedback is negative or performance takes a hit. It is this rapid delivery through CI/CD that allows Facebook to always be catering to its users and make sure they deliver valuable, working software.

This speedy delivery and continuous feedback is the driving force behind agile software development and a major objective for the Kessel Run program. Kessel Run's mission statement is to, "continuously deliver valuable software that [their] airmen love," which will only be achievable if continuous delivery becomes reality for the Air Force as it is for Facebook and other commercial software companies. Eric Schmidt, CEO of Alphabet and member of the Defense Innovation Board, stated that, "The DoD violates pretty

[8] (Rossi 2017)

much every rule in modern product development."[9] There is a major gap in capability between what modern, commercial software companies have achieved and where most DoD software programs are today. The primary objective of this project was to bridge part of that gap and increase the speed of delivery for Kessel Run, bringing the program closer to the commercial market so that developers can release and receive feedback faster, resulting in more valuable and user loved software.

7.1 External Market

Since this project is an internal service and not an external product, there is no real competition which relates to this development effort. Additionally, MITRE as a corporation is not legally allowed to compete with industry due to our charter as an FFRDC. This project is being developed by MITRE as a service for Kessel Run to use internally and will not be marketed or sold to other programs, companies, or external stakeholders.

As Kessel Run continues to grow and expand to additional Air Force programs however, there is an opportunity for knowledge sharing and reuse. In recent months, Kessel Run has begun operating as a "software factory", providing tools, services, and guidance to additional software development efforts within the Air Force. Each additional program that joins Kessel Run becomes an additional stakeholder in this project. Furthermore, other DoD programs, even outside of the Air Force, have begun looking to Kessel Run as an example. Sharing of processes and technology within the DoD is common and it is likely that the implementation of this project will be handed off to other software efforts as a reference implementation which they can adapt to their own specific use case.

7.2 Internal Market

The internal Kessel Run market benefits from this project in many ways. Due to the cross-cutting nature of the project and number of touchpoints it has with various groups, many different benefits can be identified for stakeholders. A breakdown of some of the different needs for and benefits of this project are described as follows:

Kessel Run Leadership – The business leaders of Kessel Run, including the C-Suite, have a key responsibility to quantify the value of Kessel Run's agile approach to software development. To maintain the freedoms that they have been afforded and encourage the adoption of a similar approach by other programs, leadership needs hard data showing that this approach is faster and produces better software than the traditional, waterfall strategy. To accomplish this, it is essential that they have data points around lead time between release and deployment, average release frequency, mean time to recovery, and failure rate. This project ensures that every team is tracking these four, key metrics through automation, thus allowing leadership to demonstrate the value and success of agile development.

[9] (Baram 2018)

Development Teams (Air Ops / Wing Ops) – Development teams benefit from this project because it greatly simplifies their release process and helps them ensure that their apps will work in production. Teams no longer have to manage their own version number, upload artifacts to the artifact repository, create version tags on their repositories, or alert platform operators of where to download key files from. All those steps are automatically handled by the automation developed as part of this project. Instead, teams can now focus more directly on building command and control capability and providing value to the warfighter. This project also establishes a common language and basis of understanding across the program so that as developers rotate across teams, they do not have to relearn the release process or path to production from scratch.

Security – The reputation of Kessel Run stands on its ability to produce dependable and secure applications. As part of this, each application released to production needs to comply with what's known as the Continuous Authority to Operate (CATO). The automated release process developed in this project will ensure that specific security scan requirements and criteria can be enforced prior to every application release in accordance with the CATO. Prior to this project, trust was placed on developers to manually verify that their app was meeting security requirements prior to releasing. Now, automation is in place to verify these standards, greatly improving the reliability of Kessel Run security standards.

Platform Operations – Platform operators are responsible for deploying each new version of an application into production. Before this project, that involved downloading artifacts and files from a variety of locations as specified by development teams and then communicating with the dev team product manager about how to deploy that application. It was an incredibly unreliable and time-consuming process which would not have been maintainable as more and more development teams reached their first production release. This project automated the consolidation of key artifacts and files to one location, verification that all information needed by operators is present, and deployment of applications. Removing the deployment workload from platform operators grants them much more time to dedicate to platform upgrades and maintenance instead of troubleshooting app deployments.

Platform Engineering – The platform engineering team designs a complete platform which is provided to the platform operators at each AOC. This ensures that each AOC is running an identical platform and developers have the same deployment experience regardless of which site they are deploying to. This project allows the platform engineering team to easily test new releases of the platform using the automated deployment strategy. The automated deployment also allows new instantiations of the Kessel Run platform to easily be populated with applications instead of through days of manual configuration and deployment.

Last but certainly not least is the **Warfighter**. The primary customer of all efforts within Kessel Run is the warfighter who makes use of the weapon system applications being developed to effectively run combat operations. This project enables faster updates to

those systems and in turn, drives a faster feedback cycle so the applications can better serve the warfighter.

7.3 Customer and Customer Needs Assessment

This project was developed using an agile approach in line with Kessel Run's software development methodologies (agile is covered in additional detail in section 9.1). One key concept in agile development is known as a Minimum Viable Product, or MVP. An MVP is the earliest version of a software product which addresses the most minimal set of requirements. The intent of an MVP is to produce something as quickly as possible which satisfies the initial need of the customer and can be taken back to the customer for feedback in a process known as user centered design. For this project, an MVP was developed based solely on the high-level objective of establishing a standardized release process following Google's Release Engineering principles (see section 8.2 for more detail). Through working with teams to adopt the MVP, feedback was gathered in the form of user stories.

User stories are another key tenet of agile. They provide a way to document requirements such that the intent is mapped directly back to providing user value. A good user story is written in the form[10]:

AS A persona
I WANT some feature or capability
SO THAT I can accomplish some goal
This format makes it clear to the developer exactly what needs to be done, for whom, and for what purpose without specifying exactly how it needs to be accomplished. Developers have the freedom to design the feature or capability towards providing value instead of towards constricting requirements. Once the feature is implemented, the cycle is repeated as more user feedback is collected and the design is iterated upon. A collection of user stories gathered during the rollout of the MVP for this project is given in section 8.2.

User feedback regarding the MVP was also collected through formal interviews with both platform operators and development teams. In this approach, a script is developed and used in conducting an interview with a given team. Responses and comments from the team are written down during the interview and a synth session is conducted afterwards to group the findings and identify pain points. Figure 6 below shows the interview script used in collecting feedback from platform operators and Figure 7 shows the script used in interviewing development teams. Figure 8 shows Gordon candidate Ben Reynolds during a remote interview with a development team.

[10] (Harbridge 2018)

Introduction/Overview (5 min)

1. Intros, Team intros and purpose.
2. Goals: Interview, Overview of the pipeline, Way Ahead

Interview: Understanding Current Release/Deployment Process (20 min)

1. Tell us about yourself/job/length of stay, etc.
2. Tell us about your typical day
3. What is your understanding of the Release Pipeline Process
4. What is your understanding of the Release notes/deployment strategy of dependencies, etc.
5. Is there additional information that you need and don't have?
6. Tell us about your craziest day

If they are using and have decent knowledge of release pipeline:

7. What is your understanding of the release documentation and where it lives?
8. Are there gaps in release pipeline process that you have to manually do now that could be automated
9. What are items/steps/processes of the current release pipeline that causes issues or has pain points
10. What are items you implement after the release pipeline process that are repetitive
11. What is your understanding of the Kessel Run Releases Pivotal Tracker?

Figure 6 - Platform Operator Interview Script

Introduction/Overview (5 min)

1. Intros, Team intros
2. Goals/Purpose: Interview, Way Ahead

Interview: (50 min)

1. Tell us about yourself, positions
2. Tell us about your typical day if you wish to release to production
3. How often do you release?
4. What is your understanding of the Release Pipeline Process?
5. Is there additional information that the platform operators at the site (609th/EPC) needed which wasn't sent/unavailable?
6. Tell us about your craziest day when trying to release to production
7. What is your understanding of the release documentation, notes, deployment strategy and where it lives?
8. What are items/steps/processes of the current release pipeline that cause issues or have pain points?
9. What are steps you perform manually after or before the release pipeline processes?
10. What is your understanding of the Kessel Run Releases Pivotal Tracker?
11. If you make a push to the release pipeline and if you push a sequential push where the first wasn't deployed, would you like the follow push override the first one?
12. If you had a magic wand, what would be one thing you'd change about the release process?
13. Anything else you'd like us to know or be aware of?

Figure 7 - Development Team Interview Script

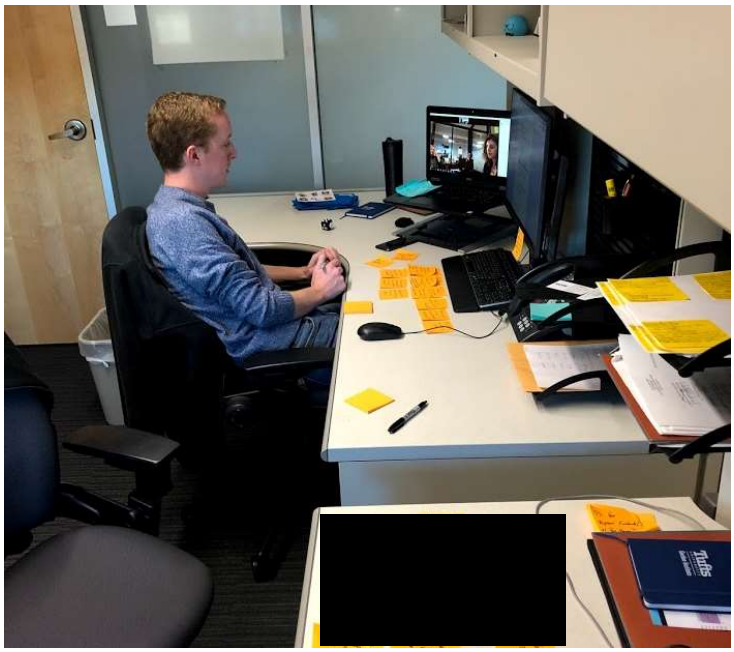


Figure 8 - Gordon Candidate Ben Reynolds Performing Interview

Figure 9 shows the result of a synth session conducted after an interview. Blue sticky notes are the comments and answers recorded during the interview session. They are grouped in this case according to what part of the release process they are most relevant to. The pink sticky notes on an angle are the pain points identified and the yellow sticky notes are follow-up questions for the group that was interviewed. Once several interview and synth sessions have been conducted, pink pain point sticky notes are correlated and prioritized for solutions.

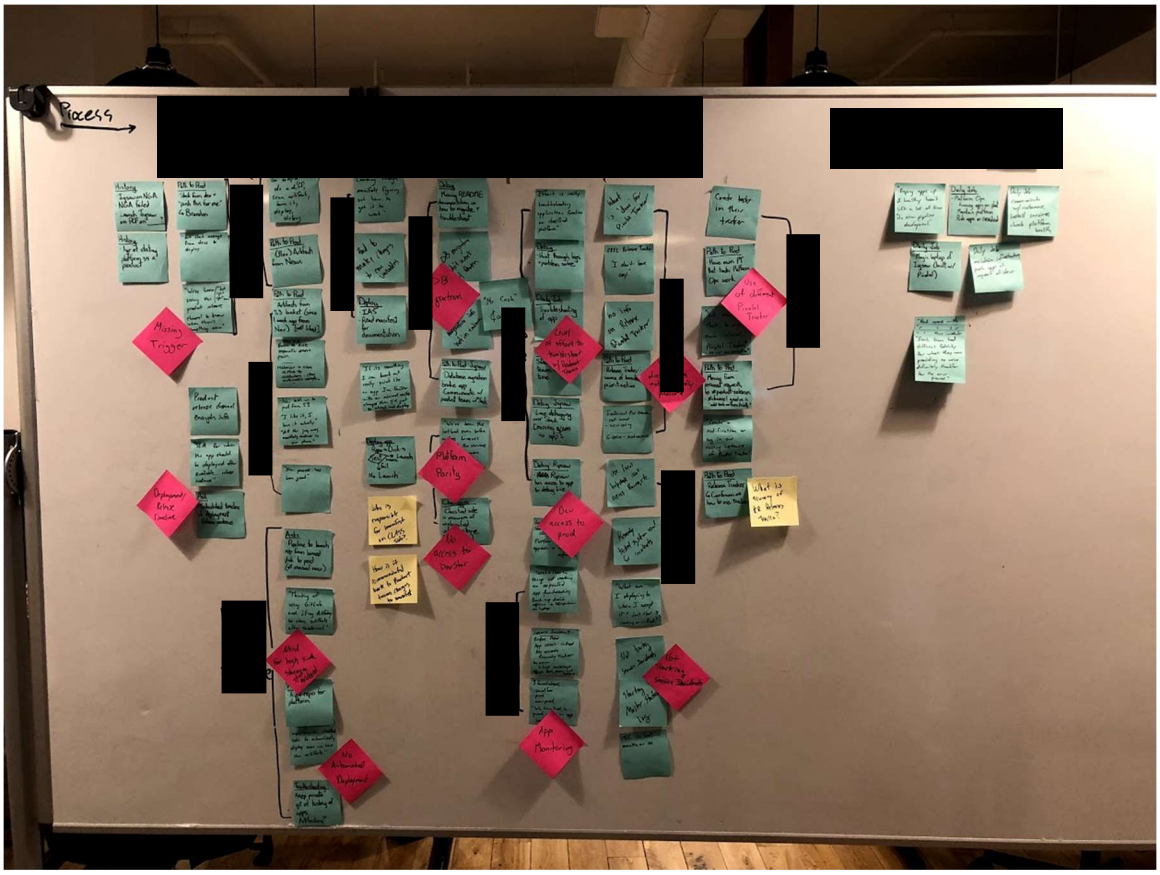


Figure 9 - Platform Operator Interview Synth

7.4 Economic Impact and Return on Investment

This project brought about tremendous value for Kessel Run through process improvement, risk reduction, and time savings. It also laid the foundation for a variety of future work which could further mature Kessel Run's software capabilities (see section 12.1).

The most tangible benefits of the project is in time saved deploying applications. In the days of 10.1 and 10.2, it could take a month or more to deploy an application into production. Using the automated deployment pipeline built in this project, an updated version of an application can be deployed in under 30 minutes. With a current count of ~20 applications teams each deploying roughly once per week to three production sites,

this puts the time savings at 93535 days / year or $35.9 million / year based on an average platform operator salary of $140,000 in Massachusetts.[11]

$$20\ [apps] \times 52\ [weeks] \times 3\ [sites] \times (30\ [days] - 30\ [mins]) = 93535\ [days]$$
$$\left(93535\ [days] \div 365\ \left[\frac{days}{year}\right]\right) \times 140{,}000 \left[\frac{\$}{year}\right] = \$35.9\ million$$

Though it isn't accurate to say that this project is truly saving 93535 days / year, what it has done is enabled the high rate of application release and delivery achieved by Kessel Run. Without projects such as this, Kessel Run may still be in a situation where an application deployment could end up taking far too long to support the fast feedback required for agile development and user centered design. This project has enabled what was previously impossible.

This project also saves time in onboarding new teams and developers through use of the example app as a starting point. Instead of having to start from complete scratch and learn all the configurations required to use Kessel Run services, teams have the ability to fork the example app as a foundation to build off. New developers can also use the example app to experiment and get familiar with the development tools. Using the following estimations, a savings of $58,915 annually can be calculated:

- 2 days saved per new team / developer
- 1 new team per month (5 personnel)
- 3 new developer hires per month
- $112,000 annual salary for java developer in Massachusetts[12]

$$8\ [devs] \times 12\ [months] \times 2\ [days] = 192\ [days]$$
$$\left(192\ [days] \div 365\ \left[\frac{days}{year}\right]\right) \times 112{,}000 \left[\frac{\$}{year}\right] = \$58{,}915$$

This project also provides many more impactful and less monetarily quantifiable benefits. One major benefit is that by creating an automated solution for release and deployment, applications can be deployed to a newly created platform with little to no effort. This means that if a new Kessel Run platform is stood up, or more importantly if one were to crash and be recreated, the entire application portfolio could be automatically deployed to it and become operational in a matter of only a few hours. This presents a huge reduction in risk and helps ensure that Kessel Run applications will always be available for use by our warfighters.

Another major benefit is that by standardizing on a process for application release, a common language and understanding has been established across the program. Developers who rotate across teams don't need to learn new processes or technologies. Everyone in Kessel Run can share the same knowledge about how applications are

[11] (ZipRecruiter n.d.)
[12] (ZipRecruiter n.d.)

delivered to production, meaning everyone is better equipped to help one another and provide value if they are reassigned to a different team.

Lastly and most significantly, this project has helped to ensure the success of Kessel Run as a program. In the DoD, it is unheard of to achieve the kind of speed of delivery that Kessel Run has for its applications and some of that success has been enabled by this project. Projects such as this one which push the envelope of DoD software development and bring new concepts to bear are critical for the transformation of government software acquisitions programs. Kessel Run is the example for how modern software development can be done in the government space and many other programs are benefitting from knowledge they've learned from Kessel Run, including knowledge of the release and deployment processes and pipelines. The 10.2 program of which Kessel Run is a successor spent over $745 million prior to its cancellation without successfully delivering a single application. Kessel Run and this project specifically, are responsible for releasing, delivering, and deploying ~20 applications currently, thus providing far more value than the $745 million spent on 10.2.

7.5 Market Challenges and Risks

The largest market risk associated with this project was that the internal market could have rejected the proposed solution. Since there really is no external market for this project, the internal customers needed to be fully bought into it to make sure that there was a target recipient. If the internal customers were to collectively determine that it had no use to them or didn't properly address their needs, the entire project concept would have needed to be reevaluated from the ground up. The secondary risk here was that the variety of internal customers present wouldn't agree on what the correct solution is. In several cases, conflicting desires from different stakeholders had to be balanced and compromises had to be made to ensure that no one group rejected the project in its entirety.

Most of the risk was addressed by getting buy-in from the Colonel who leads the C-Suite in charge of the program. The strict nature of military command means that if he supports the idea, then everyone else will have to accept it regardless of their opinion. However, this is not the desired approach and it was preferable to produce a process and product that makes each stakeholder *want* to use it. The agile development approach and consistent user feedback were integral to the process to address that exact concern. By engaging with customers / users frequently, we made sure that the process developed is one that works well for all those involved.

8 TECHNOLOGY DESCRIPTION

8.1 Overview of the Technical Challenge

As the Kessel Run program began to scale, both in number of development teams and production sites, a lack of standardized processes started to slow progress and the continuous delivery of value to Airmen. When the program began, each team was free to take their own approach to releasing and delivering their software and it was no trouble for platform operators to keep up with deploying those applications into production. The original simplicity of the apps being produced also made deployment failures a rarity with negligible impact to mission operations at the AOCs. Now however, with over 50 development teams each taking their own approach and a growing number of AOCs to deploy those applications to, the overhead for the platform operators to stay on top of the deployments would be too great, leaving them no time to focus on actual maintenance and improvement of the platform. The platform operators would have to become full time application operators: testing upgrades, troubleshooting deployment failures, and debugging development issues on behalf of developers.

Many of these issues came down to poor practice by the development teams which led to confusion or unanswered questions by the platform operators. Some examples of areas in which this was occurring are:

- Artifact versioning strategy: Different approaches to versioning meant that platform operators couldn't reliably understand what the "latest" or "previous" version of an application was.
- Artifact release repository: With some teams pushing artifacts to Nexus, some to S3 buckets, and others to the NGA (National Geospatial Intelligence Agency) platform, operators had to hunt around to locate application artifacts ready for deployment.
- Release notes: Inconsistent or entirely absent release notes means that operators weren't made aware of application changes which could affect deployment or operations strategy in production.
- Deployment configuration: Varying approaches to deployment configuration often left platform operators guessing how an application must be configured in production, leading to deployment failures and down time.

Combined, these issues painted a clear need for a standardized and automated solution to application release and deployment. IEEE Software states that, "Release engineering focuses on building a pipeline that transforms source code into an integrated, compiled, packaged, tested, and signed product that's ready for release."[13] This project would develop such a pipeline to address the concerns outlined above.

One of the major technical challenges was in devising an automated process which would be generic enough to work for all development teams with minimal impact to current

[13] (Bird, et al. 2015)

practice while also being strict enough to enforce a new and agreed upon process. The automation had to strike a balance between forcing standards on development teams and being flexible enough to accommodate a growing number of teams. Differences in build technologies used, branching strategy, repository structure, and dependencies had to all be overcome by automation. Since no two teams are the same, the automation had to be capable of spanning the differences so that individual and specifically tailored solutions weren't needed on a team to team basis. However, software is much like traditional manufacturing in that standardization can have marked benefits on sustainability, speed, and quality. Careful consideration had to be taken in deciding which of these variances were worth the up-front effort of standardizing for the sake of long-term improvement. As Dr. Robert Hall, an Indiana University professor points out, "The term "standardization" seems to connote repressed creativity - standards prevent us from doing everything in any way we choose. That is their purpose. Not all deviations are improvements. Properly used, standards are tools of progress."[14] The development teams with Kessel Run often share this negative view of standards and feel constrained when they are enforced. Like anything else though, the more purpose built the automation and process was for a single, standardized use case, the more effective it would be at accomplishing its goal. This project was designed incrementally with a goal of finding the appropriate balance to allow developers their desired freedoms while also proving effective by driving enough standards to improve practice.

Another technical challenge was to devise a solution which could scale with Kessel Run's projected growth. "Scalability is an essential component of enterprise software. Prioritizing it from the start leads to lower maintenance costs, better user experience, and higher agility."[15] Both the number of application teams and deployment sites will continue to grow at rapid pace and this solution must be capable of scaling to meet those demands without requiring overwhelming maintenance or customization. Without this scalability, the solution would quickly become useless within the Kessel Run community. This technical challenge led to the development of a configuration pipeline which greatly reduced the overhead required to maintain and operate the release pipelines.

This automation enables developers to (with minimal manual effort) release a new application to production in such a way that it also requires minimal effort from platform operators to deploy it (ideally none). The existence of a fully automated solution which begins at source code and moves all the way to a live application running in production reduces overhead for developers and operators alike, allowing more rapid and consistent value delivery to the airmen.

[14] (Hall 1986)
[15] (Conceptanext 2019)

8.2 Product Specifications

As noted in section 7.3, this project takes an agile software development approach. The initial MVP was developed in line with Google's four principles of release engineering which are outlined below with corresponding product requirements[16]:

- **Self-Service Model**
  - The pipelines shall allow development teams to release software whenever they choose.
  - The pipelines shall be fully automated.
  - The pipelines shall notify engineers if problems arise.

- **High Velocity**
  - The pipelines shall have no limit on how frequently they can be run.
  - The pipelines shall not block a development team from performing their job.

- **Hermetic Builds**
  - The pipelines must build the applications identically to how they are built by developers on their local machines.
  - The pipelines must be deterministic, behaving the same way each time they are run and providing the same result.

- **Enforcement of Policies and Procedures**
  - The pipelines must be alterable only by the release services team.
  - The pipelines must ensure that only applications which have gone through the process can be released to production.
  - The pipelines must ensure that development teams are adhering to the standard practices and policies set forth for Kessel Run.

Following the completion of the MVP, additional user stories were gathered as teams began to make use of the new process and pipelines. The collection of user stories is a continual process which aims to bring continuous improvement to the product. For that reason, a non-exhaustive list of user stories from early in the development process is provided here which demonstrate the type of requirements identified by the users.

- **AS A** developer **I WANT** a notification that my release succeeded **SO THAT** I can have confidence my release worked correctly
- **AS A** developer **I WANT** the ability to use environment variables in the artifacts file **SO THAT** I don't have to manually update the file for each release
- **AS A** release engineer **I WANT** the release pipeline to validate the artifacts file up front **SO THAT** releases fail fast and don't require manual roll-back
- **AS A** developer **I WANT** the ability to add comments in the artifacts file **SO THAT** I can make notes about the purpose of each entry

[16] (Beyer, et al. 2016)

- **AS A** release engineer **I WANT** a descriptive error message if an artifact is not found **SO THAT** I can more easily troubleshoot release pipeline failures
- **AS A** developer **I WANT** the ability to control whether my release is major, minor, or patch **SO THAT** I can release independently without needing to contact release services
- **AS A** platform operator **I WANT** the deployment pipeline to work on both unclassified internet and SIPR **SO THAT** the same process can be used for deployments to both staging and production

The requirements of this project continued to evolve as the automation matured and some of the earlier user stories were even overridden by new ones later in the project. As of April 25th, 2019, 100 versions of the release pipeline had been completed using agile iterations. Each version incorporated new user stories and requirements to continually improve the product and ensure that the automation continued to best serve the Kessel Run community. A collection of more recent user stories which reflect the greater maturity of later versions is given here:

- **AS A** security lead **I WANT** a warning message displayed to product teams when they are performing a major release **SO THAT** they are reminded to discuss architectural changes with their assessor.
- **AS A** developer **I WANT** the ability to release API stubs along with my application **SO THAT** the stubs are versioned synchronously and can be made available to other teams
- **AS A** platform operator **I WANT** automation for the creation of shared services between applications **SO THAT** I can automatically deploy applications which are co-dependent
- **AS A** data scientist **I WANT** support for the release and deployment of Docker based workloads **SO THAT** I can incorporate more complex machine learning technologies into my application
- **AS A** release engineer **I WANT** to build, scan, test, and release a single, deterministic artifact **SO THAT** there is no room for errors to be introduced in a future build of the application which is untested

8.3 Software Engineering Principles Applied

This project made use of industry standards and best practices for software engineering to ensure a high-quality, maintainable product was developed. Though these principles are typically applied to developing a software application, they can also be applied to a scripting or automation project such as this.

This first principle is modularity. Modularity implies separating software into components according to function and responsibility[17]. Within this project, there are two major pipelines, each focusing on accomplishing a specific function. The first focuses on packaging a software release and the second focuses on delivering and deploying that release. Additionally, each of those pipelines is broken down into multiple jobs (see

[17] (Shute n.d.)

section 9.3) which each address a small and deliberate unit of functionality. Breaking down the process into small, modular components makes maintainability easier as each script can be kept simple. Additionally, this modularity allows developers to more easily identify and track the steps being taken as their application flows through the pipelines.

The second principle is abstraction, which means separating the behavior of software components from their implementation. The principle of abstraction is key to this project due to its agile nature and the fact that it is provided as a service. At the start of development, it was known what the automation needed to accomplish and generally what the steps would be to accomplish it. The details of how to accomplish it (implementation) were discovered throughout the agile process however. Additionally, the underlying implementation details of the release and deployment pipelines do not need to be made known to application developers who use the process. The only thing that needs to be known by them and stable over time is the behavior. This allows the pipelines to be matured, iterated on, and adjusted without action by the development teams due to the fact that the behavior can remain constant and abstracted from the implementation.

Third is anticipation of change. Software is an automated solution to a domain specific problem. The users of the software typically have context or knowledge of the domain and the software developers learn the domain as they develop an automated solution. On this project, the application teams have context of how their applications are built and designed, platform operators have knowledge of how those applications run in production, and the automated pipelines developed here bridge the gap between those worlds. The developers on this project needed to learn both of those domains and respond to their learning by adjusting the design of the automation. For this reason, no implementation decisions were made which were hard to reverse during development or later in the future. It is best to keep the software flexible so that adjustments can be made as needed. This principle is also in line with agile, which at its core is about allowing developers the freedom to make corrections in direction as new knowledge is gained or additional insights are acquired.

Last is the concept of consistency, or the idea that it is easier to accomplish tasks in a familiar context. This principle impacts the development of this project in many ways, perhaps the most important being the selection of Concourse as the CI/CD orchestrator. Because development teams and platform operators are familiar with using Concourse in their day to day jobs currently, interacting with an automated release and delivery process built on the same technology is an easier adjustment than if a new tool were introduced. Users of the project can build on context they already have as they work to understand the new process instead of having to start from scratch. Consistency also comes into play on a more granular level on this project with regards to coding languages and standards. Bash and Python were used for this project and specific conventions for function and variable naming were adhered to so that future maintenance of the code can also take advantage of consistency. Python is one of the most popular programming languages in

use today and continues to grow in popularity over other common scripting languages like Perl, PHP, and Ruby making it a clear choice to support long term maintainability[18].

8.4 Unique Project Contribution and Intellectual Property

This project represents a totally new capability for the Air Force and a unique application of the technologies used. Though no new technologies were developed, the process and surrounding automation combined in this way represent a new capability. Under this project, a method for consistent, automated, and scalable application delivery under DoD constraints was developed, greatly pushing the envelope forward in terms of government software capabilities.

9 TECHNOLOGY APPROACH AND RESULTS

9.1 Development Approach and Methods

As outlined in section 7.3, this project was developed using agile software development strategies. The term "agile" was first popularized in "The Manifesto for Agile Software Development"[19] in February 2001. Though agile development has evolved to mean many different things and include a variety of specific practices and strategies, there are four main values that have existed since the beginning:

1. Individuals and Interactions over Processes and Tools
2. Working Software over Comprehensive Documentation
3. Customer Collaboration over Contract Negotiation
4. Responding to Change over Following a Plan

This project developed a working solution for the automated release and deployment of applications within Kessel Run following the four values given above. Each of them impacted the approach taken to development and the decisions made throughout the course of the project.

The first major decision was to develop a minimum viable product based on a minimal set of requirements. This began before a complete plan for the project was in place or before the scope of all the work to be done had even been identified. This approach is directly in line with value number four. Though the project proposal outlined a tentative project plan and schedule given in the following sections, they were subject to change throughout the development lifecycle and were adjusted based on information gathered throughout the development process. This flexibility helped to ensure that the project ended up delivering valuable software in the end. If strict plans or requirements created at the beginning of the project were followed to the letter, it is possible that the final product may not have provided any real user value. This could be due to misjudgments in the

[18] (Elizabeth 2017)
[19] (Beck, et al. 2001)

requirements writing or simply the evolution of requirements due to the speed at which software moves. Both possibilities were negated by starting from an MVP and responding to change based on user feedback throughout the process.

In order to make sure that appropriate changes were made throughout the course of this project, user feedback from developers and platform operators was collected frequently. Values one and three both confirm the need for this type of feedback. Since this project was to develop a process which makes use of tools, people needed to be at the heart of the design. Without a focus on usability and enabling efficient interactions between individuals and teams, the automated process developed through this project would have been ineffective. The development teams and platform operators are more important than any tool, technology, or process and the development approach taken in delivering this solution reflected that. As the end users (customers) of this project, development teams and platform operators were invited to collaborate at every step of the process through interviews (section 7.3), paired programming, and technical design meetings.

Lastly, this project placed a higher emphasis on producing working software than thorough documentation. In particular, one of the primary vehicles for documenting this project is a fully functional example application which development teams can reference when they have questions about how the process works. The example application implements all of the requirements to be in line with the new release and delivery process developed during this project and evolved in parallel with the automated pipelines. The pipelines themselves have corresponding documentation but rely heavily on the example offered by the sample application. The code for the project is also thoroughly commented such that it is self-documenting and requires less external documentation. The external documentation is primarily focused on high level objectives and points of interaction between different individuals in the process (value number one).

9.2 Testing, Verification and Validation Implementation

Several forms of continuous testing occurred throughout the development of this project. The first and primary means of testing was to perform a release of the example application any time code changes were made in the automation. This can be considered “functional testing” since it involved a true execution of the product. Since the example app is representative of the applications built in Kessel Run and meets all the requirements to be supported by the release and deployment pipelines, pushing it through the process would serve to catch most bugs. This type of testing was enabled by the approach taken to versioning the release pipeline. Configuration files for each application allow release engineers to set the version of the automation which is active for a given team. This prevents updates to the automation from “going live” until they have been tested using the example application. In some cases, if bugs were identified by a specific application team, fixes would also be tested against the application which triggered the bug in the first place.

The second type of testing performed was unit testing. An approach to testing the scripts used in the pipeline was developed allowing release engineers to confirm that changes

made to one part of the code did not break existing code. This type of unit testing provides very fast feedback but is limited in what it can catch. Unit testing is intended to catch bugs such as syntax errors, typos, or issues with logical structure whereas the functional testing described above verified overall functionality. Unit testing was used most heavily in testing the "releasebot" which developers use to trigger a release. Each change to the releasebot passed through its own continuous integration pipeline which would run a full suite of unit tests prior to deploying the update. Figure 10 below shows output of the continuous integration pipeline for releasebot running unit tests and reporting on code coverage.

```
----------------------------------------------------------------------
Ran 8 tests in 0.026s

OK
Name                                          Stmts   Miss  Cover   Missing
---------------------------------------------------------------------------
app/__init__.py                                   2      0   100%
app/blueprints.py                                16     16     0%   1-22
app/es/__init__.py                                7      2    71%   8-9
app/es/constants.py                               2      0   100%
app/es/documents/ReleaseRequest.py               19      2    89%   24-25
app/es/documents/__init__.py                      0      0   100%
app/gitlab/__init__.py                           12      6    50%   10-15
app/google_sheets/__init__.py                    13      2    85%   16, 20
app/lunch/__init__.py                             1      1     0%   1
app/lunch/event_handler.py                       11     11     0%   1-20
app/responders/__init__.py                        4      0   100%
app/responders/base.py                           13      1    92%   17
app/responders/default.py                         8      0   100%
app/responders/lunch.py                           9      0   100%
app/responders/lunch_selection_strategy.py       18      0   100%
app/responders/release.py                        88      8    91%   10-13, 16-17, 20-21, 84, 124
app/slack/__init__.py                             9      0   100%
app/slack/message_handler.py                     18      0   100%
---------------------------------------------------------------------------
TOTAL                                           250     49    80%
```

Figure 10 - Releasebot Test Coverage

Lastly, but most importantly, the release automation underwent continuous user testing due to the agile development approach. After being tested internally through unit testing and use of the example app, new versions of the pipeline would be rolled out incrementally across the application teams. The automation was architected in such a way that release engineers could perform these updates without any action on behalf of the development teams and without incurring any down time. Then, as those teams performed releases, they would of course be testing the update. The benefit of this approach is two-fold. For one, this allows release engineers to confirm that the product is fully working for real Kessel Run applications and development teams. Secondly, this provides opportunity for a closer relationship between the release services team and the development teams, in turn providing a better user feedback loop and user centered design.

9.3 Results and Technical Conclusion

Though an agile project of this nature is never truly “complete”, the result of this project is a mature solution for application release, delivery, and deployment within Kessel Run. Over 100 iterations were done on the release pipeline which is at the heart of this project, taking it through five major revisions. As the release pipeline matured, so too did the corresponding deployment pipeline and the supporting tooling such as the releasebot and config pipeline. The following three figures show the progression of the release pipeline through its five major revisions and the relative increase in complexity and capability at each major release.

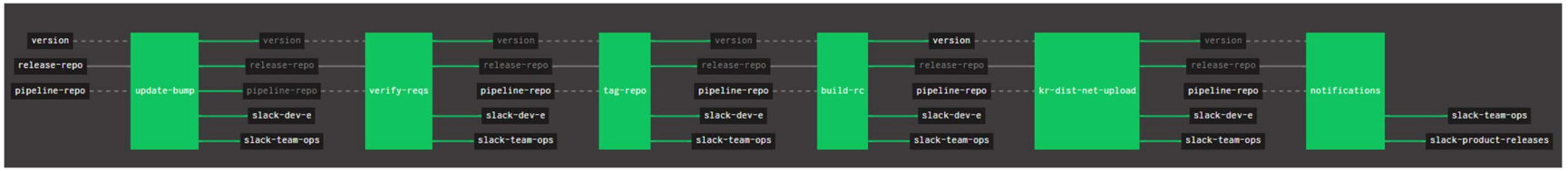


Figure 11 - Release Pipeline Version 1,2,3

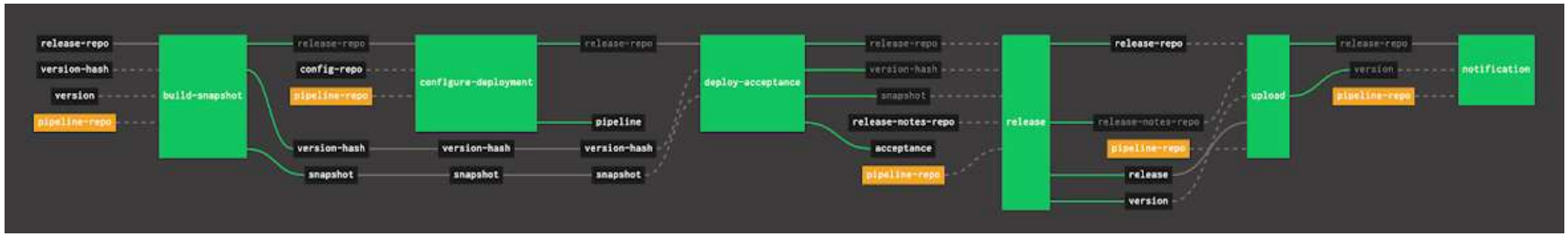


Figure 12 - Release Pipeline Version 4

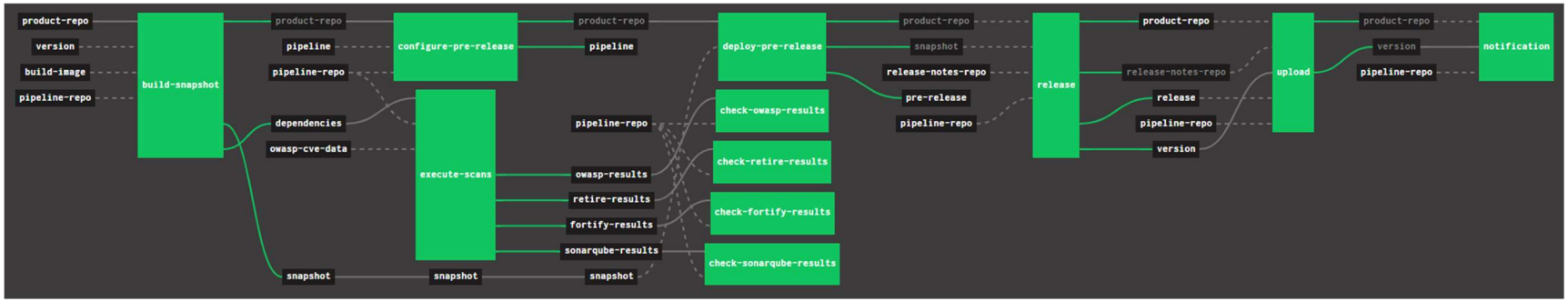


Figure 13 - Release Pipeline Version 5

In release pipeline versions 1 through 3, each source code commit pushed to the release pipeline would trigger a release. The pipeline would then take the following steps to perform a release:

1. Update the pipeline (see section 9.4)
2. Verify the presence of required files
3. Tag the source code repository with the updated version number
4. Build the application artifact (jar, war, zip)
5. Create and upload a release bundle to an S3 bucket for distribution
6. Send success notifications

This solution was simple and effective but had two major pain points which were revealed via user interviews with application development teams. The first was that the commit which was pushed to trigger the release needed to contain a specific snippet of the form "[bump: <major|minor|patch>]" which informed the release pipeline how to increment the version number for that release. This caused developers to frequently have to make empty commits containing this special snippet in order to retrigger the pipeline in the event of a failure. It also made it challenging for developers to push code to the release pipeline programmatically from their own CI pipelines. The second major issue was that this pipeline would build the artifact from source and release it without providing the developers a chance to verify the built artifact. Developers wanted a way to double check that the artifact built correctly prior to it being released.

Version 4 was built specifically to address and improve upon those two major pain points from the first three versions. It also marks the single biggest jump in the capabilities of this project. While versions 1 through 3 were refactors of an existing code base, the team started over with version 4 to ensure it was implemented in the most ideal imaginable at the time. When release pipeline v4 was triggered by a commit, the following steps would be taken:

1. Build the application artifact and upload it as a snapshot
2. Configure the automated service provisioning (see section 9.4)
3. Deploy the application to a "pre-release" environment for verification
4. Wait for a release request from a developer in slack
5. Increment and tag the version number and promote the snapshot to a release artifact
6. Create and upload the release bundle to S3 for distribution
7. Send success notifications

The deployment to a pre-release environment provides two major benefits. For one, developers and product managers can access that environment to manually verify their application is working as expected prior to releasing it. Secondly, the deployment logic is in perfect parity with that of the deployment pipeline used on SIPR. This pre-release deployment serves as a test of the environment configuration and deployment automation prior to application release. The introduction of releasebot also provided dual benefit. Since a manual release trigger was introduced, developers can push to this pipeline as frequently as they want without actually performing a release. The releasebot also took

control of managing the version bump (major, minor, or patch) so that it didn't have to be included in the commit message. Under release pipeline version 4+, developers need only send a message to releasebot which states "release <app-name> as <major|minor|patch>". They will then be prompted for a confirmation and the release will be triggered to deliver whichever build of their app is currently in the pre-release environment. Figure 14 below shows this interaction occurring for a release of the example "hello-world" application.

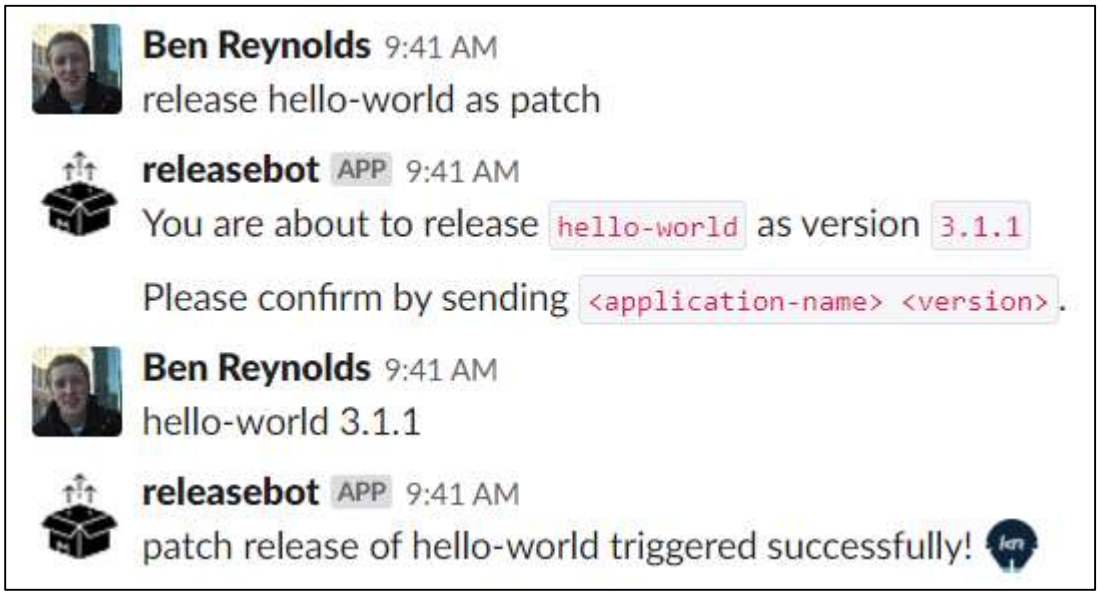


Figure 14 - Releasebot Interaction

The current major version of the release pipeline, version 5, was introduced to address the concerns of security stakeholders. Prior to version 5, there was no software in place to strictly enforce that applications released through the pipeline had passed security scanning criteria. As shown in Figure 13 above, release pipeline version 5 became a combined security and release pipeline. Developers could now view a single pipeline for all of their security scan results as well as their release automation. Additionally, the releasebot was updated to verify that the code being released meets the security criteria prior to releasing. This allows the release services team to confirm with much greater certainty that any application being released to production is safe for operational use on the classified network.

As of April 30, 2019, there have been 294 successful releases of Kessel Run applications using the release pipeline. Figure 15 below shows the high-level path those releases take to get to production including the release/security pipeline and the deployment pipeline.

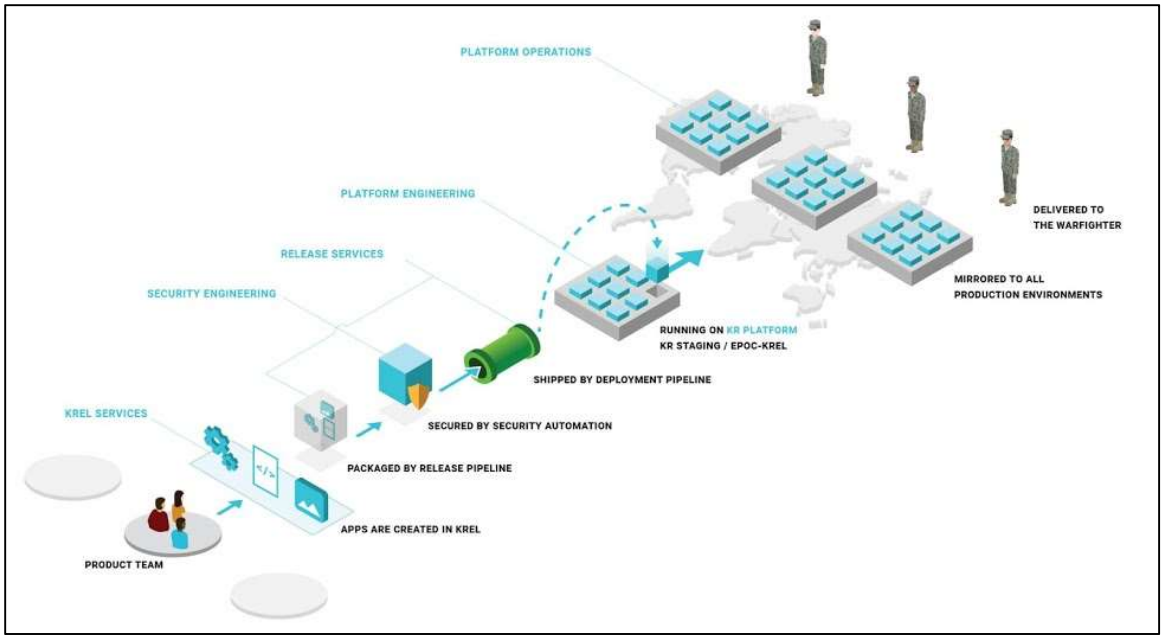


Figure 15 - Path to Production

Despite the release pipeline being the central focus of the project, the overall project architecture has grown substantially throughout development in order to support the introduction of more features and capabilities. The project architecture in use as of April 30, 2019 is provided in Figure 16 below.

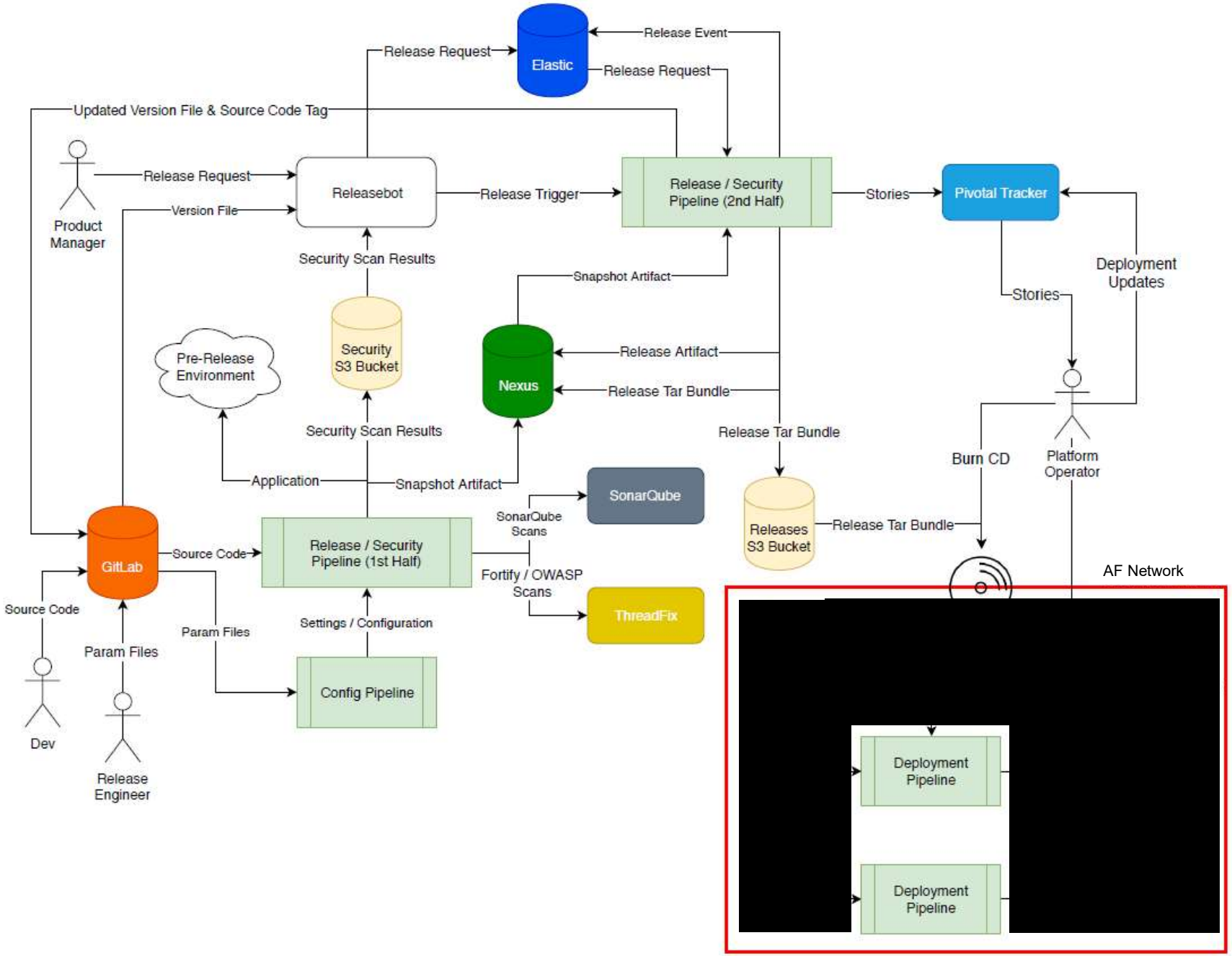


Figure 16 - Release and Deployment Architecture

9.4 Scientific and Technical Challenges

Perhaps the biggest, unexpected challenge faced during the development of this project was due to a limitation of Concourse. In Concourse, the structure of a pipeline is defined as a single yaml file which when uploaded to Concourse informs which steps will be taken and in what order. This file includes many runtime parameters and unfortunately, cannot easily be adjusted from within the pipeline itself. Concourse is designed such that the steps and parameters for a pipeline need to be known at the time it is created and remain unchanged as the pipeline runs.

For this project, the pipelines needed the ability to adjust based on the input they were given. To avoid making a specifically tailored pipeline for each application, the generic pipeline had to be able to read from the source code repository and take appropriate action based on that input. There were two primary use cases in which this type of behavior was necessary:

1. Controlling Version Increment – The release pipeline versions applications using semantic versioning which has the form X.Y.Z. X represents a major release which includes breaking changes; Y is a minor release with backwards compatible improvements or feature additions; Z is a patch release with backwards compatible bug fixes. Each release via the pipeline needed to be able to specify whether it was a major, minor, or patch release but this parameter is set in the main yaml file.

2. Automated Service Provisioning – The release and deployment pipelines both have the ability to automatically provision middleware services such as databases, message queues, OAuth connectors, etc. prior to deploying an application into the environment. The steps to perform these actions, however, are more examples of actions which would normally be preset in the main yaml file. In this use case, the pipelines needed to read service definition files from a source code repository and correctly provision the needed services. This is due to the fact that not all applications use the same services or provision them using the same configuration.

To overcome this limitation, the release and deployment pipelines each have a job which reads in the required information, generates an updated yaml file for itself, and resets its own configuration in Concourse. In this way, each pipeline is essentially constructing itself during runtime and can dynamically adjust based on input. This type of behavior falls outside of the intended use case for Concourse and required significant and complex scripting to accomplish in a reliable manner. Currently, there is an open issue against Concourse in GitHub to provide this type of functionality out of the box which could greatly simplify the pipeline implementations in the future.[20]

The second unforeseen technical challenge was to devise a solution for managing all of the release pipelines. With such a small team, manually setting, updating, and maintaining release pipelines for 20+ applications amounted to an unacceptable amount

[20] (Suraci 2016)

of overhead. Any time an update to the pipeline was made, each release pipeline would need to be manually "reflown" or reset in Concourse. Care had to be taken to ensure that only the teams intended were updated as well. With the speed at which this project was progressing, this update process often had to be repeated several times a week at a cost of about half an hour each.

The solution to this challenge was to develop a supporting pipeline known as the config pipeline. Each of the configuration files for an application's release pipeline was placed in a single repository. These files contain information such as the team name, product name, important urls, target deployment environments, and most importantly the version of the release pipeline which should be active for that application. The config pipeline was designed so that whenever a release engineer makes changes to one of those configuration files and pushes it to GitLab, the config pipeline automatically kicks-off and updates the release pipeline controlled by that file. This allows release engineers to track all of the configuration as code and make declarative changes to release pipelines which are automatically applied. It also makes bulk updating the pipelines (such as putting all application teams on the latest release pipeline version) much easier. A release engineer can simply open the config repository in their editor, search and replace all the release pipelines versions, and push the committed changes to GitLab. This type of workflow is sometimes referred to as "GitOps". Desired state is stored as configuration files in a source code repository (config files) and automation is used to ensure that state is maintained across an environment (release pipelines).[21]

10 PROJECT PLAN

10.1 Statement of work

The purpose of this project was to bring release engineering and continuous delivery principles from industry to bear in Kessel Run. This project helps Kessel Run achieve its goal, which is to continuously deliver valuable software that airmen love. By creating an automated and standardized approach to the release and deployment of applications into production, workload on development teams and platform operators was reduced, thus increasing time spent on value delivery and increasing the speed of development.

The following assumptions were made in asserting that this project could successfully accomplish the scope of work set forth on time and on budget:

- AWS Diode will be the cross-domain solution procured for Kessel Run
- Tools used for automated deployment must be capable of receiving Authority to Operate
- Concourse must be used for automation in the unclassified development environment
- Pivotal Cloud Foundry will be the sole deployment platform
- Adoption of the automated process will be enforced by Kessel Run leadership

[21] (Riley 2018)

10.2 Schedule

10.2.1 Work Breakdown Structure

Figure 17 below shows the original work breakdown structure as laid out in the project proposal. The work would be organized around three major components: the release pipeline, the deployment pipeline, and the example application. Nearly all of the work given in this WBS was completed as planned with the exception of item 2.4 which was later deemed to be lower priority than other features encountered through the agile process. The agile process did however open much opportunity for more complex advancements which led to the project having a larger scope than originally planned. The updated WBS given in Figure 18 accurately reflects the high-level tasks which were accomplished during this project. Many of the additional tasks added during the course of the project were in support of accomplishing the original tasks more effectively. The releasebot and config pipeline are both examples of major tracks of work and separate products which were added later in the development process to serve in producing a better release and deployment process.

Level 1	Level 2	Level 3
1 Release Pipeline	1.1 Minimum Viable Product	1.1.1 Version Tracking / Updating
		1.1.2 Repository Tagging
		1.1.3 Artifact Build & Upload
		1.1.4 S3 Artifact Consolidation
		1.1.5 Notifications & Pivotal Tracker
	1.2 MVP Adoption	1.2.1 Pre-MVP Teams
		1.2.2 Post-MVP Teams
	1.3 Release Pipeline V2	1.3.1 Security Pipeline Integration
		1.3.2 Additional Feature from Users
2 Deployment Pipeline	2.1 S3 to ECS Synchronization	
	2.2 Service Creation	2.2.1 Service Definition Plan
		2.2.2 Automated Service Provisioning
	2.3 Application Deployment	
	2.4 Release Acceptance	2.4.1 Automated Smoke Test
		2.4.2 Manual Release Approval
3 Example Project & Documentation	3.1 Example Application	
	3.2 Example CI Pipeline	
	3.3 Release Pipeline Documentation	
	3.4 Path to Prod Workflow Documentation	

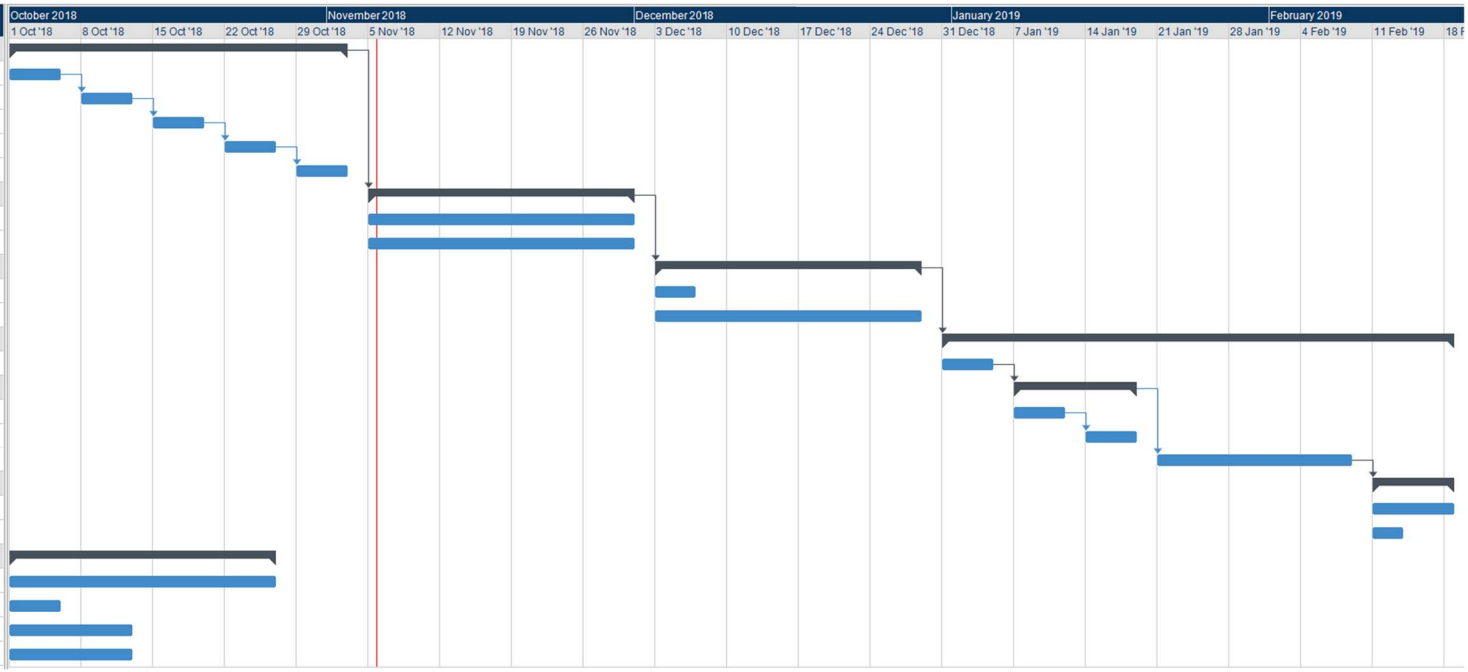
Figure 17 - Original Work Breakdown Structure

Level 1	Level 2	Level 3
1 Release Pipeline	1.1 Minimum Viable Product	1.1.1 Version Tracking / Updating
		1.1.2 Repository Tagging
		1.1.3 Artifact Build & Upload
		1.1.4 S3 Artifact Consolidation
		1.1.5 Notifications & Pivotal Tracker
	1.2 Release Pipeline Adoption	
	1.3 Release Pipeline Version 3.X	1.3.1 Deployment Pipeline Integration
		1.3.2 Artifacts File
		1.3.3 Version Increment Control
	1.4 Release Pipeline Version 4.X	1.4.1 Pre-Release Deployment
		1.4.2 Releasebot Integration
		1.4.3 Snapshot to Release Promotion
		1.4.4 Metrics Integration
	1.5 Release Pipeline Version 5.X	1.5.1 Security Integration
		1.5.2 Deployment to Unclass Production
2 Deployment Pipeline	2.1 S3 to ECS Synchronization	
	2.2 Service Creation	2.2.1 Service Definition Plan
		2.2.2 Automated Service Provisioning
	2.3 Application Deployment	
3 Example Project & Documentation	3.1 Example Application	
	3.2 Example CI Pipeline	
	3.3 Release Pipeline Documentation	
	3.4 Path to Prod Workflow Documentation	
4 Releasebot	4.1 Manual Release Requesting	
	4.2 Security Integration & Blocking	
5 Config Pipeline		

Figure 18 - Updated Work Breakdown Structure

10.2.2 Gantt chart

Task Name	Duration	Start	End
Minimum Viable Project	**25 days**	**10/1/2018**	**11/2/2018**
Version Tracking / Updating	5 days	10/1/2018	10/5/2018
Repository Tagging	5 days	10/8/2018	10/12/2018
Artifact Build & Upload	5 days	10/15/2018	10/19/2018
S3 Artifact Consolidation	5 days	10/22/2018	10/26/2018
Notifications & Pivotal Tra...	5 days	10/29/2018	11/2/2018
MVP Adoption	**20 days**	**11/5/2018**	**11/30/2018**
Pre-MVP Teams	20 days	11/5/2018	11/30/2018
Post-MVP Teams	20 days	11/5/2018	11/30/2018
Release Pipeline V2	**20 days**	**12/3/2018**	**12/28/2018**
Security Pipeline Integration	4 days	12/3/2018	12/6/2018
Additional Features from...	20 days	12/3/2018	12/28/2018
Deployment Pipeline	**36 days**	**12/31/2018**	**2/18/2019**
S3 to ECS Sync	5 days	12/31/2018	1/4/2019
Service Creation	**10 days**	**1/7/2019**	**1/18/2019**
Service Definition Plan	5 days	1/7/2019	1/11/2019
Automated Service Provis...	5 days	1/14/2019	1/18/2019
Application Deployment	15 days	1/21/2019	2/8/2019
Release Acceptance	**6 days**	**2/11/2019**	**2/18/2019**
Automated Smoke Test	6 days	2/11/2019	2/18/2019
Manual Release Approval	3 days	2/11/2019	2/13/2019
Example Project / Documen...	**20 days**	**10/1/2018**	**10/26/2018**
Example Application	20 days	10/1/2018	10/26/2018
Example CI Pipeline	5 days	10/1/2018	10/5/2018
Release Pipeline Document...	10 days	10/1/2018	10/12/2018
Path to Prod Workflow Doc...	10 days	10/1/2018	10/12/2018



10.2.3 Project Planning Assessment

There were many changes to the original plan throughout the course of this project. Some of that change was due to technical discoveries made along the way. Other was due to the introduction of new stakeholders or organizational changes. In both cases, the flexibility and adaptability provided by the agile approach was able to handle the change and keep progress moving on the project.

One of the significant changes which occurred near the end of the project was a decision by the Kessel Run Enterprise Services branch chief to combine the release engineering and security engineering teams. She recognized the desire of our stakeholders to make security a first-class citizen from an application release and delivery perspective and decided that combining the two teams would be one of the most effective ways to make that happen. This change was also due in part to the high-level of success that had been achieved with the release and deployment pipelines and a desire to bring the security pipeline to a similar point.

The original plan was to consider this project "complete" after the delivery of release pipeline version 4 (see section 9.3). However, this unforeseen change in the organizational structure and in stakeholder priority led the newly combined team to have to create version 5. This was totally new territory for engineers on the prior release engineering team who now had to learn the security domain and vice versa for the security engineers who had to learn release. There was also significant technical challenge in producing a combined pipeline which would provide fast feedback to development teams on their security posture while also ensuring the integrity of applications being released.

The team was able to tackle these challenges effectively due to smart design decisions made earlier in the development of the release pipeline and config pipeline for managing it. The solution was addressed in multiple phases (as with any agile project). First was to place the source code for the security pipeline in the same repository as the release pipeline. This allowed the config pipeline to take ownership of managing both the release and security pipelines. Secondly, the security pipeline jobs were updated so they would be able to integrate with the release pipeline. The security pipelines for all the application teams were then updated to this new version to test compatibility. Once compatibility with the updated security pipeline was confirmed, the security jobs were added to the release pipeline to produce release pipeline version 5 as shown in Figure 13 (affectionately known as the "secrete" [security-release] pipeline). The end result of this change was a pipeline that better addressed stakeholder needs, worked better for the development teams who no longer had to reference two separate pipelines, and made better use of engineering resources to build and manage these pipelines.

Throughout this change, one of the things learned by the Gordon candidate is that it is good to have strong opinions loosely held. Having strong technical opinions typically comes from experience and in this case, Ben was the most experienced automation developer on the combined release and security team. He had many firm opinions about

how integrating the two pipelines should work and how best to tackle the problem. However, those opinions had to be loosely held to avoid conflict with the new team members from security. They as the developers of the security pipeline and with a different stakeholder perspective also had strong opinions. If both sides had stuck to their strong opinions and held them tightly, little to no progress would have been made in addressing the challenges ahead. It was only through discussion and compromise from both sides that a solution was able to be found.

Overall, this project accomplished everything it set out to originally and far more. The original scope was to automate application release and deployment and build an example application which could be used to demonstrate and document the process for development teams. In the end, a complex system-of-systems for application release, security, and delivery at scale was produced and continues to drive Kessel Run's capabilities forward. This project team has continually coped with change to produce a valuable product and is referred to as "one of the highest performing engineering teams in Kessel Run" by the Enterprise Services branch chief.

10.3 Budget and Costs Assessment

The only costs incurred to produce this project were labor costs for the engineers on the team. As shown in the Gantt chart in section 10.2.2, the original time allocation was 4.5 months to complete the planned work. The proposal laid out a budget based on this schedule and a full-time staff of three engineers. Labor costs are estimated here based on an annual salary of $140,000, the average for platform operators (similar skillset) in Massachusetts.[22] Actual figures for staff expenses are proprietary information. Given a full-time staff of three individuals over 4.5 months, the total estimated cost of this project comes to $337,500 as shown in the following equation:

$$\frac{\$140{,}000}{12 months} \times 4.5 months \times 3\ people = \$157{,}500$$

This original estimate was nearly exact. Release pipeline version 4 was released on February 11, 2019, marking all of the original work laid out in the proposal as complete one week ahead of the originally projected end date of February 18, 2019. This work was completed using the three full-time engineers originally planned as laid out in the proposal.

However, the scope of this project grew as previously discussed. The introduction of additional requirements and the merger with the security team gave this project additional engineers and more work. On top of the 3 engineers over 4.5 months, additional work completed to date added an additional 1.5 months with original staffing of 3 and one additional month with an expanded staff of 6. This brings the final project cost to $280,000 or 177% of the original budget.

[22] (ZipRecruiter n.d.)

$$\frac{\$140{,}000}{12 months} \times ((6 months \times 3\ people) + (1\ month \times 6\ people)) = \$280{,}000$$

One thing learned by the Gordon candidate as a result of this budget phenomenon is that good work is rewarded with more work. The budget ballooned due to an increase in project scope which stemmed from the success of the base implementation. If the project were to have missed the mark in accomplishing what it originally set out to, the opposite would have happened and future development would have ceased immediately. Instead, Kessel Run leadership recognized the potential this project has to continue driving impactful change and opted to increase its staffing allocation and continue the work.

10.4 Risk Plan and Mitigation Assessment

The risk assessment laid out in the proposal is provided below in Table 2 for reference. Of the risks originally identified, the only one which occurred was that the cross-domain solution was not in place by the end of the project. This risk was successfully mitigated using the actions planned and the release and deployment process have been used very successfully despite having a manual transfer between classification levels. All of the other mitigation actions listed were also taken during the course of this project with marked success. For example, one identified risk was that program churn and restructuring may deprioritize the project. By keeping leadership well informed of the successes and impact of the project, the opposite took place and the Enterprise Services division was restructured in an effort to further prioritize the project and its continued impact.

Risk	Potential Impact	Likelihood	Actions to Mitigate
On-boarding teams to MVP may be more intensive and time consuming than anticipated.	Moderate	High	• Provide prep-work action items to teams to speed up the on-boarding process • Address more simple teams first to work out kinks before working with complex teams
Buy in from teams may be low causing a rejection of the new process or resistance to adopt	Moderate	Moderate	• Hold process introduction meeting to explain reason behind change and value proposition for teams
Cross domain solution may not be	Low	High	• Design process such that it can be

in place by end of project timeline			used without having cross domain solution in place
Providing support and maintenance for release pipeline may detract from development of deployment pipeline	Low	Moderate	• Build release pipeline to be as self-service as possible • Implement descriptive error messages with corrective actions
Program churn and restructuring may deprioritize the project	High	Low	• Make sure importance and value is communicated to Kessel Run leadership • Keep leadership informed of early successes
Low staffing may lead to siloed knowledge around implementation which could deteriorate value if current team members relocate in the future	High	Low	• Actively share knowledge with individuals throughout Kessel Run • Provide enough documentation that someone totally new could carry on with the work

Table 2 - Initial Risk Assessment

11 LEADERSHIP

11.1 Leadership Capabilities Assessment

At the beginning of the project, Ben identified "Communicating and Advocacy" and "Realizing the Vision" as his two leadership capabilities most in need of improvement. Figure 19 on page 53 shows the leadership capabilities spider chart self-assessment performed by the candidate at the beginning of the project. As shown, communicating and advocacy along with realizing the vision are the two lowest scores on the chart. Throughout the course of the project, Ben had many opportunities to exercise not just these, but nearly all of the capabilities listed on the chart. As a result of this project, he has become a much more effective leader as well as a more confident engineer.

At the onset of the project, Ben considered his skills in communicating and advocacy to be in need of improvement. In his career up until that point, he had generally failed to advocate for himself and his ideas and to convince others that his ideas were worth following. The entire success of this project hinged on his ability to do just that, however. Throughout the entire development of the project, Ben routinely met with key stakeholders to make sure they were well informed of progress and incremental victories. These interactions also served as opportunities to receive important feedback and adjust the direction of the project as needed. These stakeholders spanned many different teams including platform engineering and operations, application development across multiple portfolios, security, metrics, and the C-Suite.

Perhaps the most challenging of these groups which pushed the candidate's communication and advocacy capabilities to improve the most was the development teams. The primary focus of the application development teams in Kessel Run is to build software which provides value to the warfighter. Every action they take is weighted against how well it will help to accomplish that objective. For this reason, those teams typically viewed release pipeline work as non-value added tasks (known as chores) and would push back if they perceived the workload to be too great or a detractor from their objective. These teams also represent the primary consumer of the project, however, and their buy-in was essential. To overcome this challenge, Ben and Tom Cashavelly (ISA and PM, see section 11.2) set up bi-weekly meetings with portfolio leadership for the application development branches. These meetings consisted of progress updates, discussions of project benefits, and an opportunity for the stakeholders to ask clarifying questions and provide feedback. Through these meetings, Ben was able to convince the portfolio leaders of the value of this project and in turn, influence the development teams. Ben also spent a significant amount of time pairing along side developers to advocate for the project directly. By working with them in accomplishing the on-boarding work and sacrificing his own time to support their advancement, he demonstrated a commitment to the value of the project and helped the developers to see it as well. In the end, every application team in Kessel Run has been on-boarded to the release and deployment

pipelines built as part of this project and the project has become a key capability for the program as the official method for software delivery.

The other area in which Ben grew tremendously over the course of this project was realizing the vision. This project really represents the first opportunity the candidate had to take a complex, engineering project from inception to completion. In doing so, he demonstrated the ability to overcome barriers to success and ensure a valuable and impactful product was delivered to Kessel Run. At the beginning of the project, Ben had a vision, although rather fuzzy, of what this project could be. He recognized the potential for impact and wide scale change and understood that if done correctly, this project would impact not just Kessel Run but DoD software development at large.

One of the challenges in making this happen was to balance technical implementation with the surrounding soft skill tasks. If the project were implemented perfectly and worked 100% of the time but nobody knew about it or used it, the vision would have fallen flat. Alternatively, if everyone was bought into the idea of automated application delivery but nothing was ever built, the vision would've collapsed for the entirely opposite reason. The Gordon candidate demonstrated a ruthless commitment to success in ensuring that both of these outcomes were prevented. He demonstrated technical excellence in designing and coding the implementation as well as dedication to the program in helping to support the project's adoption.

Though communicating and advocacy and realizing the vision were the two leadership skills most improved during the project, the other capabilities improved as well. A listing of several other leadership capabilities and brief descriptions of the opportunities Ben used to improve them is provided here:

- Initiative – When the opportunity to increase the scope and impact of this project presented itself, Ben repeatedly seized the opportunity and took advantage of agile development to incorporate new capabilities into the project.

- Decision Making – Ben routinely had to weigh multiple potential solutions against one another and decide on a way forward. Though he didn't get this right every time, the implementation decisions he made resulted in a highly successful product for Kessel Run.

- Courage – Bringing this project to success required on several occasions that Ben take a risk on his own capabilities to advance progress. Whether it was affirming that a feature request would be possible or trusting his instinct enough to push back against opposition, Ben had to stand up for himself and trust that he would be able to deliver on his promises.

- Inquiry – Through stakeholder interviews and assisting development teams, Ben received extensive practice in learning how to ask the right questions. In order to extract the necessary information to make informed design decisions, he had to

carefully craft his questions and ask lots of them. Assuming his own expertise and failing to ask clarifying questions would have quickly driven this project to fail.

- Interpersonal Skills – Kessel Run is a rapidly growing organization and this project impacts nearly every person on the program. As the project lead, Ben interacted with hundreds of individuals during development and ended up making friends with many of them. By treating people with respect and getting to know them, he was also able to more effectively influence them towards desired outcomes.

- Connect Across Disciplines, Skills, and Cultures – As already mentioned, this project impacted nearly everyone in Kessel Run including developers, security, leadership, product managers, platform operators in other countries, etc. To effectively design and implement with each of these groups in mind, Ben had to connect with each of them. He had to understand the young government civilians writing Java code and the experienced contractors operating the platform on the other side of the globe. Throughout this project, he had engagements with people of many backgrounds, cultures, and especially disciplines and learned how to adjust his leadership style to suit the situation.

This project also opened the candidate's eyes to his reliance on expert power to gain influence in the workplace. Ben is considered the subject matter expert on automated pipelines for all of Kessel Run and tends towards using this expertise as a primary means of influencing. In combination with referent power, this proves an effective tool. Typically, Ben's first engagements with individuals on the program are to provide technical support. He gains trust through providing sound recommendations, solving complex problems for others, and assisting in a broad spectrum of technical challenges. Once that trust baseline is established, he can build the relationships on a more personal level and gain additional referent power as well. This use of power has earned him a seat in many important meetings for Kessel Run making decisions that impact the technical direction of the program.

Overall, Ben considers himself to be a much stronger leader now than at the time of the project proposal. His self-assessment scores at the end of the project clearly show this improvement and growth. In particular, the significantly higher scores indicate more than anything else that Ben is a more confident leader. This project has proven to him that he

does have the ability to effectively lead a team and drive project success. The end of project spider chart is shown below in Figure 20 for comparison purposes.

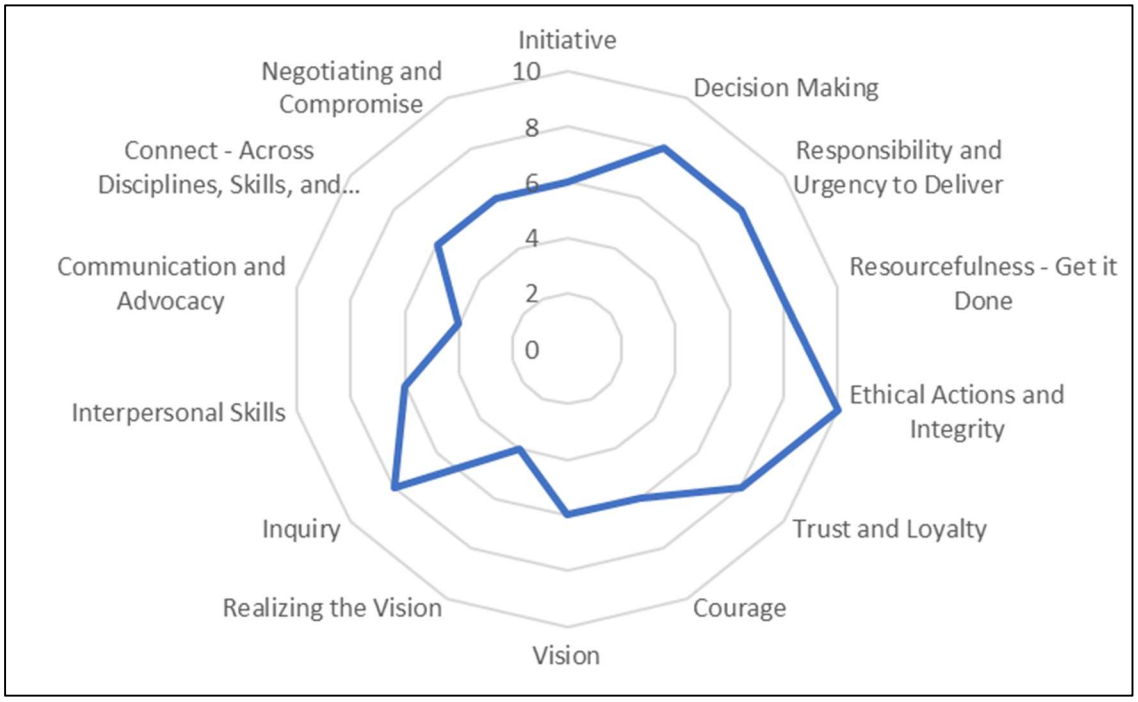


Figure 19 - Leadership Capabilities Self-Assessment (Project Start)

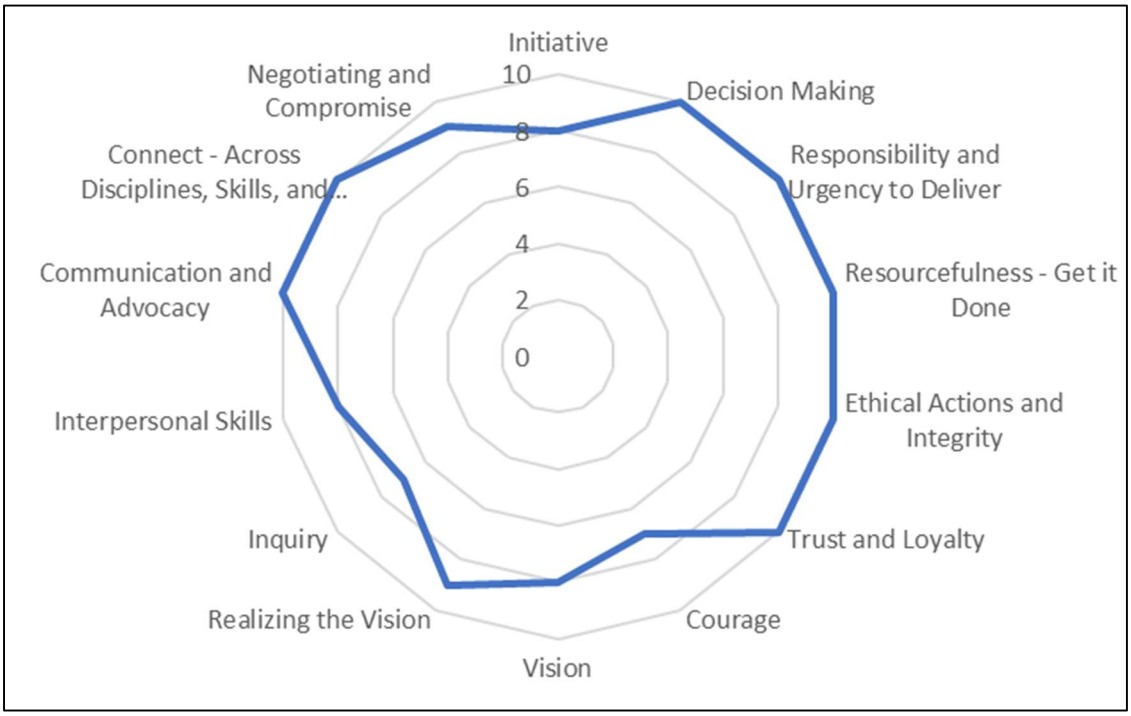


Figure 20 - Leadership Capabilities Self-Assessment (Project End)

11.2 Team staffing and organization

Gordon Candidate: Benjamin Reynolds
ISA: Thomas Cashavelly
Gordon Mentor: Jane Eisenhauer
Faculty Advisor: David Kaeli
Others: Micah Algard, Caroline Lee, Josh Tassone, Eric Bram

Figure 21 below shows the team structure for this project. The Gordon Project Team was composed of four individuals. The Gordon Candidate, Ben Reynolds, was the team lead responsible for the technical and managerial success of the project. Ben's background is described in section 2. The Industry Sponsor Advocate (ISA) was Thomas Cashavelly who has approximately eight years of experience with MITRE. He has held such roles as a software engineer, technical lab lead, task lead, group lead, and product manager. Thomas also completed the Gordon Engineering Leadership Program at Northeastern. Jane Eisenhauer, the Gordon Mentor, is also a Gordon Fellow of Northeastern University and currently holds a lead position at Raytheon. Her role was to assist the Gordon Candidate in keeping the project on track and succeeding in producing a quality challenge project. Finally, the Faculty Advisor, David Kaeli, has been an advisor to Gordon Candidates for over ten years and brings extensive knowledge of both the program and software engineering. He was Faculty Advisor to Thomas Cashavelly when he went through the program as well. His role was to ensure the technical rigor of the project and provide technical guidance to the Gordon Candidate.

As noted in section 10.3, the technical project team within Kessel Run expanded from its original size at the onset of the project. Unchanged were the roles of the Gordon candidate, Ben Reynolds, and ISA, Tom Cashavelly. The remaining team structure was impacted as a result of the merger with the security engineering team during project development. Caroline Lee was the lead engineer on the security engineering team prior to the merger with release engineering. She has approximately three years of experience at MITRE and a strong background in cyber security making her a very valuable asset to the team. She and Ben now serve as co-anchors (engineering leads) on the combined team with Ben focusing primarily on the technical design and implementation and Caroline focusing on making sure security objectives are well met. Micah Algard served as a developer on several application teams prior to joining release services. He joined the team as this project was starting and did paired programming with the Gordon candidate until he gained the necessary experience to act as an individual contributor. He now specializes in the release and deployment aspects and works closely with application teams to manage the configuration of their deployment environments and automated deployment. Eric Bram and Josh Tassone were both engineers and the security team prior to the merge. As some of the original contributors to the security pipeline, they each bring a strong technical background and domain knowledge regarding the security scanning and authority to operate. They now develop as a pair and focus primarily on improving the security components of the pipeline as well as working closely with product teams to manage onboarding to the release / security pipeline.

As the most experienced member of the team with the technologies at play, the Gordon candidate, Ben Reynolds, had many opportunities to lead and mentor this team throughout the project. From the beginning, Ben mentored Micah through paired programming and taught him all of the required domain knowledge as well as the use of Concourse, Python, and many other supporting technologies. He then performed extensive enablement after the merger with the security team in order to bring their engineers up to speed and help improve the efficiency of the combined team and product. Ben also provided the vision and guidance across the team as discussed in greater detail in section 11.1.

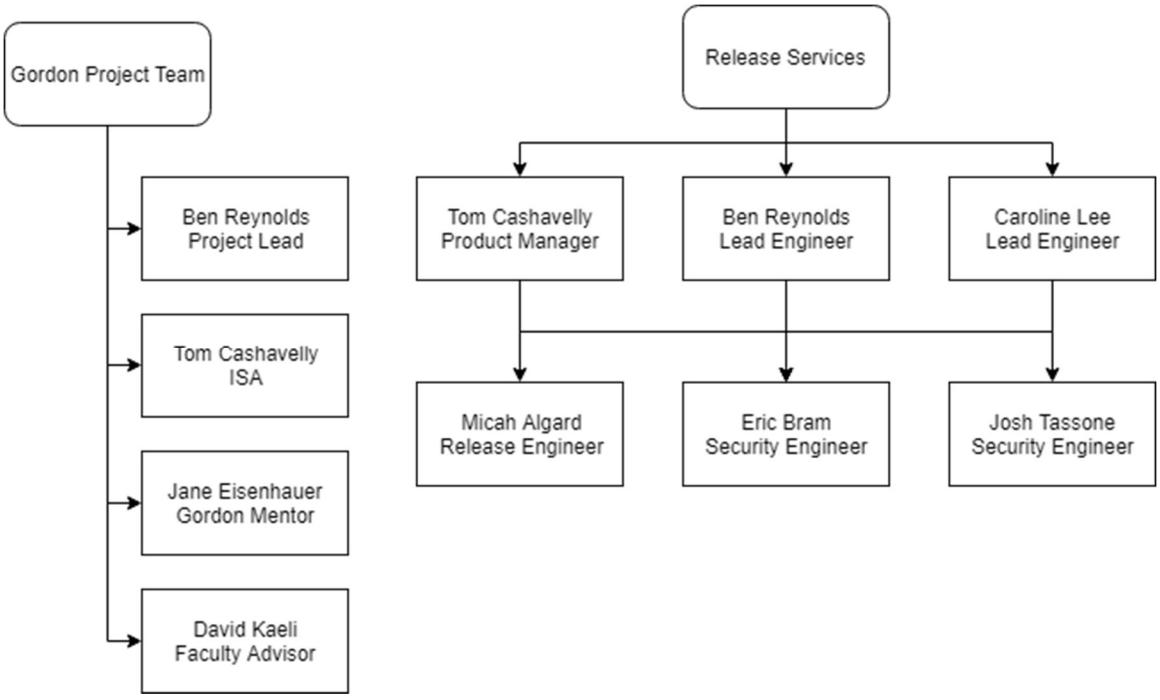


Figure 21 - Project Team Structure

12 SUMMARY

This challenge project has been an incredibly valuable experience for the Gordon candidate, Ben Reynolds. This project has been his first opportunity to take a project from inception to completion, to have a leadership role on a team, and to truly showcase the full extent of his abilities as an engineer. This work has earned him recognition both within Kessel Run as well as within MITRE.

At Kessel Run, Ben has become widely acknowledged as a leading expert on automated pipelines and application delivery. He is frequently consulted on important matters and gets much more opportunity to impact the program at a larger scale than before the project began. In addition, the release engineering team which he was instrumental in starting and is now an engineering lead on has been referred to by the branch chief as "one of the highest performing engineering teams in the organization."

Within MITRE, Ben has had the opportunity to share knowledge from this project with many other DevOps engineers across the company. He has given a presentation open to the whole company on technical implementation and strategy and provided follow-on consultations for employees on several other programs spanning even beyond the defense sector. His contributions to the Air Operations Center project within MITRE (under which Kessel Run falls) have also been recognized with a Catalyst Award for becoming "an exceptional technical leader", a Spark Award, and a Program Recognition Award, MITRE's highest program honor.

Ben is incredibly thankful the opportunity to undergo this project, for the help and mentorship he has received along the way, and for the growth he has experienced as a result. It is his hope and expectation that this personal improvement will only continue to accelerate his career growth and open countless opportunities to provide even greater value and impact in the future.

12.1 Recommendations for Future Work

This project has opened the door for a wide variety of follow-on efforts. Currently, this project only supports the release and deployment of "single artifact applications" which can be deployed to Pivotal Cloud Foundry with a simple "cf push" command. One major area for future exploration is support for releasing and deploying Docker based workloads. This is a feature that is in Kessel Run's platform roadmap and Dockerized applications will soon need to become a fully supported deployment mechanism.

Another major work area is to improve the flexibility provided in application deployments. The automation as it is built now is designed to deploy the latest release of a single application to a single platform. Though this automation can be replicated through multiple instantiations for multiple apps or multiple platform environments, it does not allow application developers to specify their global deployment strategy. For example, if developers wanted a new version of an app available at several sites for testing or training and an older version or versions(s) deployed elsewhere, they should be able to specify that deployment topology via a manifest or configuration file which is consumed by automation. One tool which may be able to provide this type of capability is known as Spinnaker[23] and should be investigated further for its potential use on Kessel Run.

The last area which should be explored further is a cross-domain solution for pushing application releases from the unclassified development environment to the classified network for operational deployment. Right now, the requirement of manually transferring media via compact disc greatly slows down the application delivery process. Though research has already begun in this area, it has the potentially to totally revolutionize the way Kessel Run and other software programs deliver software and should be considered an utmost priority. Only with an effective cross-domain solution in place will Kessel Run ever reach true continuous deployment and achieve the fast feedback loop and deployment speed desired.

[23] (Netflix n.d.)

13 REFERENCES

Baram, Marcus. 2018. *The U.S. Air Force learned to code—and saved the Pentagon millions.* July 5. Accessed March 31, 2019. https://www.fastcompany.com/40588729/the-air-force-learned-to-code-and-saved-the-pentagon-millions.

Beck, Kent, Mike Beedle, Arie Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, et al. 2001. *Manifesto for Agile Software Development.* Accessed April 24, 2019. https://agilemanifesto.org.

Beyer, Betsy, Chris Jones, Jennifer Petoff, and Niall Richard Murphy. 2016. *Site Reliability Engineering: How Google Runs Production Systems.* O'Reilly Media, Inc. https://books.google.com/books/about/Site_Reliability_Engineering.html?hl=&id=_4rPCwAAQBAJ.

Bird, Christian, Tamara Marshall-Keim, Bram Adams, Foutse Khomh, Kim Moir, and Stephany Bollomo. 2015. "The Practice and Future of Release Engineering." *IEEE Software.*

Conceptanext. 2019. *The Importance of Scalability In Software Design.* February 21. Accessed April 29, 2019. https://conceptainc.com/blog/importance-of-scalability-in-software-design/.

Curtis E Lemay Center For Doctrine Development and Education. 2016. "Executing Operations." November 22. Accessed May 12, 2019. https://www.doctrine.af.mil/Portals/61/documents/Volume_3/V3-D13-Executing-Ops.pdf.

Elizabeth, Jane. 2017. *Python ascending: Where have all the scripting languages gone?* November 13. Accessed April 24, 2019. https://jaxenter.com/tiobe-index-scripting-languages-138905.html.

Hall, Dr. Robert. 1986. "Continuous Improvement Through Standardization." *Essentials of Excellence.*

Harbridge, Teagan. 2018. *How to Write Good User Stories in Agile Software Development.* March 15. Accessed April 29, 2019. https://blog.easyagile.com/how-to-write-good-user-stories-in-agile-software-development-d4b25356b604.

Netflix. n.d. *Spinnaker - Continuous Deployment for Enterprise.* Accessed May 1, 2019. https://www.spinnaker.io/.

Potter, Matthew. 2009. *Why is Defense Acquisition So Difficult.* June 24. Accessed May 12, 2019. https://www.cbsnews.com/news/why-is-defense-acquisition-so-difficult/.

Riley, Chris. 2018. *GitOps 101: What Is GitOps, and Why Would You Use It?* August 6. Accessed April 30, 2019. https://www.twistlock.com/2018/08/06/gitops-101-gitops-use/.

Rossi, Chuck. 2017. *Rapid release at massive scale - Facebook Code.* August 31. Accessed March 31, 2019. https://code.fb.com/web/rapid-release-at-massive-scale/.

Shute, Gary. n.d. *Principles of Software Engineering.* Accessed April 24, 2019. https://www.d.umn.edu/~gshute/softeng/principles.html.

Suraci, Alex. 2016. *Dynamic Build Plan Generation.* October 6. Accessed April 30, 2019. https://github.com/concourse/concourse/issues/684.

The MITRE Corporation. 2013. *Our History.* May 18. Accessed March 31, 2019. https://www.mitre.org/about/our-history.

United States Air Force. 2017. *Combined Air Operations Center (CAOC).* July 1. Accessed June 3, 2019. https://www.afcent.af.mil/About/Fact-Sheets/Display/Article/217803/combined-air-operations-center-caoc/.

ZipRecruiter. n.d. *What Is the Average Java Developer Salary by State.* Accessed April 29, 2019. https://www.ziprecruiter.com/Salaries/What-Is-the-Average-Java-Developer-Salary-by-State.

—. n.d. *What is the Average Platform Engineer Salary by State.* Accessed April 29, 2019. https://www.ziprecruiter.com/Salaries/What-Is-the-Average-Platform-Engineer-Salary-by-State.

14 SIGNATURE PAGE