Chaos Engineering for Cloud Resiliency

Report Prepared By:

Renee Sawka, Nestor Lopez, and Olivia Scola

Chaos Engineering for Cloud Resiliency

A Major Qualifying Project
Submitted to the Faculty of
WORCESTER POLYTECHNIC INSTITUTE
in partial fulfillment of the requirements for the
degree of Bachelor of Science

Project Team
Renee Sawka rcsawka@wpi.edu
Nestor Lopez nllopez@wpi.edu
Olivia Scola ogscola@wpi.edu

Project Advisor
Professor Wilson Wong
Department of Computer Science

Project Co-Advisor
Professor Robert Sarnie
WPI Business School

Date Submitted: 13 December 2022

Abstract

Partnering with State Street to investigate strategies for cloud engineering, the project team created a scalable automation framework that facilitates the design of resilient services. The process leverages Azure DevOps pipelines to provision cloud resources and subjects them to failures such as outages, access denials, and CPU overloads, producing data that exposes vulnerabilities in enterprise systems. In an age where cloud migration is emerging as a competitive necessity in industry, this framework enables chaos engineers to reinforce their internal architecture against critical and unforeseen threats.

Executive Summary

Throughout the last decade, State Street has begun overhauling its application infrastructure, migrating critical services to the cloud for the many benefits it offers. Despite the inherent resilience and agility of cloud services, it remains important for these applications to be designed and implemented properly to meet business goals. Even though systems are carefully planned with this elasticity in mind, misconfiguration and implementation gaps could create catastrophic incidents resulting in financial and reputational risk for the firm. To expose and correct for these gaps, State Street needed a way to routinely subject their services to possible failures to ensure the resiliency of their services in the cloud, a process known as chaos engineering.

Azure is a cloud provider that offers many resources and services such as virtual machines, virtual networks, Kubernetes and data storage solutions. Along with these cloud services, Microsoft is developing Azure Chaos Studio, a platform where chaos engineers can create experiments that expose resources to outages, overloads, and other system failures in order to uncover flaws in the design of their services. Microsoft has also developed Azure DevOps Pipelines, a service that manages routine jobs for automated testing and deployment and is capable of interacting with the Azure resource manager.

Over the course of seven weeks, the project team endeavored to develop a framework for State Street's cloud application testing. Utilizing agile scrum methodology and modeling principles of chaos engineering, the objective of this work was to leverage Azure Chaos Studio to produce a useful tool. During the first half, the team familiarized themselves with State Street's infrastructure, Azure cloud networking, DevOps pipelines, cloud resources, and chaos experiments. During the second half, the team utilized their new understanding to develop scripts to conduct routine chaos experiments and effectively outline a replicable procedure that can be used to run these failure scenarios on other company resources and applications.

The documentation the project team provided to State Street will support future development and automation of chaos engineering. As State Street expands into new segments, the knowledge gained through the project will enable rapid cloud application deployment, shortening their product-to-market time and ensuring the reliability of their services. With State Street's overarching corporate aim of providing comprehensive and secure financial services, this project works to achieve this ideal by helping engineers maintain the highest standards of agility, quality, and resiliency.

Acknowledgements

This project would not have been possible without the support and guidance of many individuals and organizations. We would first like to thank our advisors, Professor Wong and Professor Sarnie, for their indispensable feedback and unwavering guidance. Their constant support throughout our weekly meetings was a large positive contributor to the outcome of this project. We would also like to extend our thanks to Professor Blais for his help leading up to this term as well as his additional support throughout our project.

We would especially like to thank State Street Corporation for their willingness and excitement to partner with WPI and our project group to pursue this initiative. A special thank you specifically to Ravi Palanki and Prafull Srivastava, our primary sponsors and points of contact, for their passion and efforts towards this project and for connecting us with relevant individuals in the company. Additionally, we would like to thank Akshay Sharma for coordinating this project with our advisors and for the input he gave to shape the initial direction of our work.

We would also like to express our gratitude towards all the individuals at State Street who we were introduced to during our time with the company that went above and beyond in helping us overcome the obstacles we encountered. Thank you to Anurag Phadke, Swapnil Puri, Purush Rudrakshala, Ranjeet Garodia, Yerrabati Sudheer Kumar, and Monica Kashyap for all of their varied assistance with troubleshooting Terraform and the establishment of our Azure DevOps pipeline repository. Thank you also to Pawad Sood, Gagandeep Singh Saluja, Sumith Ayyappankutty, and Praju Varma Ravi Varman on the team working on AKS Chaos Engineering for their assistance with our Kubernetes experimentation. We sincerely appreciate the cooperation and enthusiasm of all the team members, as well as the valuable insights they provided towards this broader company initiative.

Thank you to the developers who contributed to Azure, Terraform, Terragrunt, Kubernetes, Python, Docker, and any contributors of the Linux project because without these tools our work would not be possible. We thank the developers who published Terraform modules for us to freely use and modify, as well as the developers who published tutorials for Azure Chaos Studio and other Azure services.

Finally, we would like to thank the individuals who published their learnings to the internet through YouTube, Wikipedia, Stack Overflow, and other free knowledge bases, because without these resources we would not be able to learn the many concepts required to complete this project in time. Thank you.

Table of Contents

List of Figures

List of Tables

Authorship

Section	Main Author(s)	Main Editor(s)
Abstract	All	All
Executive Summary	All	All
Acknowledgements	Sawka	Lopez
Chapter 1. Introduction	Lopez	Sawka
Chapter 2. Research		
2.1 Company Background	Sawka	Scola
2.2 Background Research		
2.2.1 The Financial Incentive of the Cloud	Lopez	Scola, Sawka
2.2.2 Security	Lopez	Scola, Sawka
2.2.3 Industry Standards	Lopez	Sawka
2.2.4 Cloud Service Delivery	Sawka	Sawka
2.2.5 Chaos Engineering	Sawka	Sawka
2.2.6 DevOps	Lopez	Sawka
2.2.7 The Infrastructure as Code Paradigm	Sawka	Sawka
2.3 Business Risk Considerations	Scola, Sawka	Scola, Sawka
Chapter 3. Software Development Methodology	Sawka	Scola
Chapter 4. Software Development Environment		
Introduction Paragraph	Sawka	Lopez
4.1 Project Management Software	Sawka	Scola
4.2 Integrated Development Environment	Sawka	Scola
4.3 Additional Software Tools		
Introduction Paragraph	Sawka	Scola
4.3.1 Azure Resource Manager	Sawka	Lopez
4.3.2 Azure Chaos Studio	Lopez	Sawka
4.3.3 Azure Command Line Interface	Lopez	Sawka

4.3.4 Terraform	Lopez	Sawka
4.3.5 Kubernetes		
4.3.6 Zoom, WebEx	Sawka	Scola
4.3.7 Microsoft Teams, Microsoft Outlook, iOS Messaging		
Chapter 5. Software Requirements	Sawka, Lopez	
Chapter 6. Design	Lopez, Sawka	Scola, Sawka
Chapter 7. Software Development	Sawka	Scola
Chapter 8. Assessment		
8.1 Business Learnings	Scola	Sawka
8.2 Technical Learnings	Lopez	Sawka
8.3 Accomplishments		
8.4 Limitations	Sawka	Lopez
Chapter 9. Future Work	Sawka, Lopez	
Chapter 10. Conclusion	All	
Appendix A: Glossary of Terms	Sawka	
Appendix B: Gantt Chart	Scola	Sawka
Appendix C: Developer Story Point Burndown	Sawka	
Appendix D: MVP Proposal	Lopez, Sawka	
Appendix E: MVP Deliverable		

1. Introduction

Enterprise data, once penciled into a filing cabinet, now flows through a labyrinth of corporate firewalls, virtual networks, and fiber-optic cables. Business operations now execute in an immense digital frontier of virtual machines, operating on the processors of the colossal cloud data centers of the twenty-first century. By commodifying processing power at a competitive price, these computational powerhouses are forcing companies to abandon their basement-dwelling mainframe computers and migrate their data and services to the cloud.

Cloud migration is not just an interesting alternative to mainframes, it is a financial necessity for many companies. The added cost-efficiency and reliability of a properly configured cloud platform is imperative for many service providers if they want to survive in a competitive market (Avram 2014). The increasingly unnecessary round-the-clock maintenance of private servers is not only a financial burden, but also poses an existential threat to companies with competitors that have already migrated their services to the cloud. This threat will only worsen as cloud data centers improve their technology and increase cost efficiency. This forced exodus of corporate services makes cloud migration strategies and tools extremely time sensitive.

As imperative as this transition to the cloud is, this is not a process that can be rushed since data breaches can have catastrophic effects on the economy and national security (Aday 2019). Since the new location of these services requires them to be accessed solely through the internet, migrating companies must prepare to defend against the myriad of malicious actors who are eager to exploit the inevitable holes in the soup of rapidly evolving cloud software. As more conglomerates managing trillions of dollars of assets transfer critical economic operations to this unprecedented technological environment, it becomes absolutely necessary that their services remain reliable and secure.

Prior to the computing revolution, migrating corporate data between physical locations may have only required a scenic stroll with a briefcase. In today's age migrating natively digital enterprise services is no walk in the park. Careful consideration of software dependencies, hardware architectures, and network configurations must be taken to ensure proprietary software does not encounter compatibility issues with its environment. Physical cables must now be converted into virtual networks, leveraging the tools of cloud service providers to preserve reliable communication between clients and services. Operating system specifications, once installed on the company mainframe, have to be containerized into virtual machine images capable of interfacing with the cloud provider’s resource management software. Above all, these corporations, who once secured their internal services with physical walls, now require a whole new level of security.

Fortunately, despite all the challenges, the formidable task of cloud migration can still be accomplished with the many available tools and guidelines to help companies perform the adoption. Organizations like the Cloud Security Alliance or CSA research and publish best practices on how to secure this daunting multifaceted environment. One of the CSA's latest publications, the fourth version of the Cloud Controls Matrix or CCM, cements the expectations for cybersecurity controls. This document details and categorizes about two hundred named security responsibilities, each complete with their respective corporate divisions and appropriate implementation guidelines to secure the technical components involved (Cloud Security Alliance 2021). The Cloud Security Alliance, along with similar organizations, establish concrete guidelines that clarify the necessity of secure cloud migration. Alongside this literature, the industry has provided "Infrastructure as Code" or IaC tools to provision and maintain resources across multiple cloud providers. Independent of the multitude of cloud control interfaces, IaC tools allow concise definitions of the storage, network, and computing resources needed to replicate the services' pre-migration environments.

State Street is a major financial technology firm to say the least. Offering investment servicing, asset management, and financial data analytics, their business relies on efficient processing of information. With trillions of dollars worth of assets under management, State Street cannot cut any corners when it comes to the security and reliability of their systems. They play a key role in the global economy and as the world of FinTech shifts to the cloud, State Street has a responsibility to not only stay current with market trends but to trailblaze the development of emerging technologies. While they have been experimenting with the Microsoft Azure Cloud in recent years, State Street currently hosts the majority of their internal and client-facing services in a private data center.

Problem

"State Street has embarked on a journey to modernize the critical infrastructures with a cloud-first strategy, which spans across multiple cloud providers. As we are transforming and leveraging the agility, scalability, and resiliency of the cloud, we are taking on new responsibilities per the shared responsibility model of a cloud provider" (State Street Azure Platform Engineering 2022). State Street is currently in the process of migrating many of their services to Microsoft Azure Cloud data centers. They have a hard deadline to accomplish this, and many services that were hosted in their private data center must be safely established in the cloud by this time. These services must be resilient to system failures and maintain the same availability as before. Cloud migration is a complex process, especially for proprietary software designed for a mainframe environment, but if they do not meet this deadline, there will be broader business implications.

Goal

"To improve the resiliency of applications deployed on cloud, we are planning to establish chaos engineering practices to uncover such problems early during the cycle. For the initial phase, the plan is to start with Chaos Studio on the Azure Cloud environment" (State

Street Azure Platform Engineering 2022). State Street has tasked the project team with establishing a chaos engineering framework. This will be a set of documents and scripts that will leverage Azure DevOps pipelines to provision cloud resources and subject them to Azure Chaos Studio experiments, enabling developers to automate routine chaos scenarios such as service outages and network segmentation. The framework will allow these experiments to be configured alongside the IaC, and make it easy to run custom data collection scripts in parallel. This framework will enable State Street's migration team to ensure the strength of their cloud services.

2. Research

2.1 Company Background

Headquartered in Boston, Massachusetts, State Street Corporation is the largest custodian bank in the world, a financial services and bank holding company providing investment management, trading research, and financial analytics for investors (Parsons 2018). Operating through two primary lines of business, State Street provides settlement and payment services through their investment servicing division and strategies and products through their investment management division. With roots back to the 1700s, State Street found its beginnings as Union Bank before later becoming a national bank in the 1800s (Chcom 2020). Following a 1925 merger with an American mutual fund custodian, the company began operating under its present-day name and continued to grow throughout the decades before eventually forming Boston Financial Data Services in 1973 (Lyudvig 2022). With IBM engineering spearheading the technological development of the company, the turn of the 21st-century is when State Street really started to put the 'tech' into FinTech. Selling their retail and commercial banking businesses in 1999, State Street solidified themselves as a securities processing and asset management company (Lyudvig 2022). This led to their 2018 acquisition of Charles River Development, a major provider of investment management software, and with their newly acquired technology and expertise, State Street was now suited to create proprietary software (Haque 2018). In 2020, State Street launched its Alpha data platform, a cloud-based investment servicing software that has helped the company establish itself as a leader in digital assets (Strack 2022). State Street has also been making headlines for their other advancements in digital technology as they look to tokenize funds and private assets (Strack 2022). This ability to utilize technology to meet the needs of customers has allowed the company to establish corporate stability in the financial sector. This sentiment is reflected in State Street's purpose, which is "To help create better outcomes for the world's investors and the people they serve" (O'Hanley 2022). With a long and proven history of delivering comprehensive and flexible financial solutions, State Street services help investors achieve their business objectives worldwide.

2.2 Background Research

2.2.1 The Financial Incentive of the Cloud

From the end user's perspective, enterprise services are just web pages displayed on their screen. In reality, these services rely on a number of fine-tuned systems in order to handle the daunting volume of modern internet traffic (IBISWorld 2022). As an incoming request packet enters the data center's walls, it is routed through a labyrinth of cables and switches, loaded into server memory cells, and processed by the tangled transistors of sizzling hot silicon. Immediately, the server conducts the formation of a well-written response that is sent through the network to the end user. As suggested, this intricate dance of advanced electrical machinery is not exactly easy to choreograph.

In a traditional data center, maintaining reliable connections between these systems involves the management of physical cables, network switches, and computing devices that must be routinely inspected and replaced to ensure 100 percent connectivity (Barroso 2013). Due to the constant demand for their services, these computers must always stay online, generating heat that must be dealt with by reliable cooling systems. These systems are expensive to buy, difficult to install, and draw an immense amount of electricity once operational (Barroso 2013). Adding to the monthly electricity rates, a data center must pay internet service providers a premium for fast, high bandwidth connections if they want their services to satisfy end users. All these systems need physical space to occupy, IT specialists to maintain them, and security guards to defend them, which demands a massive investment just to begin operations (Barroso 2013). Before the cloud, every company that wanted to host a professional web service needed to perform this herculean feat.

Cloud computing data centers have been established to take on this financial and managerial burden, selling computing and networking benefits at a bargain. Companies no longer need to pay technicians to fix their servers or security guards to defend their networks. While only needing to pay for the computing resource usage without the initial down payment, the cloud has made hosting enterprise services much more affordable. Even conglomerates with established physical data centers for their services pay more than necessary because cloud data centers systematically delegate computing tasks among processors, often ensuring more efficient use of electricity and time (Kaur 2017). Every computation done on outdated corporate data centers results in a loss of capital, forcing these enterprises to consider selling the data centers and migrating to the cloud. For a startup business in today's world, building a personal data center is financially unthinkable when their competition does not need to. This revolutionary business of selling resources has disrupted the market and its prices have created an irresistible incentive to move enterprise services to the cloud.

2.2.2 Security

Cybersecurity was once far more straightforward than it is today. The typical enterprise computer of the twentieth century was a mainframe used for data processing and storage on magnetic tape, hosted within the company’s walls and only accessible from inside (Arzoomanian 2009). A brick wall separated the hackers from any software vulnerabilities, requiring physical infiltration of the company headquarters. While data breaches still happened, they were much more complex and could always be thwarted by age-old practices like locking the door. When the first online customer services like product catalogs and staff directories came about, hackers had a point of entry through the internet, giving rise to Denial of Service and SQL injection attacks (Horner 2017). Even though hackers had more to play with, simple cybersecurity practices could thwart most of these attacks. Since administrative system access could still only be done entirely within the company walls, the damage a hacker could do from outside those walls remained limited. If a malicious action or accident did manage to cause a system outage, responding technicians had direct access to the troubled servers and could yank the power cables themselves if needed.

When enterprise services are moved to the cloud, the physical security of their data is now the cloud provider's responsibility and the migrating company can only hope that the data center's walls are properly guarded. The easing of physical security responsibilities often comes as a relief since the migrator's role in the cybersecurity realm has just gotten much more complicated. Administrative duties involving any form of software updates or system control must now be conducted remotely, requiring careful management of encryption keys for each employee that needs access (Cloud Security Alliance 2021). As services become more complicated, more administrators are needed, leaving more places for hackers to find valid authentication keys to grant them remote privileged access to cloud services. With many employees now working from home, these authentication keys may be scattered across personal computers, only increasing the likelihood that these keys fall into the wrong hands (Barrero 2021). With privileged access to enterprise systems, malicious actors can cause anything from service outages to devastating leaks of personal information.

There are several practices that engineers can employ when designing their cloud environments to mitigate vulnerabilities. In cases where companies desire many networks to divide the scope of services, they can create a hub-spoke network topology where many application-specific "spoke" networks are peered with a single "hub" network (Microsoft 2022). One benefit of this topology is that public internet access controls can be centralized to the hub, requiring only a single internet firewall for all spoke networks to use. With internet access reduced to a single point, firewall configuration is easy to manage. Another security practice involves limiting the images that can be used when creating virtual machines or VMs. By only allowing hardened images hosted on private image galleries, VMs may only be started with pre-inspected software and fitted with improved security, such as antivirus and internal firewall configurations. While maintaining their own set of VM images takes work, it allows migration teams to mitigate software vulnerabilities and place controls on the initial states of virtual machines. Another security practice involves protection of secret information. Many cloud providers offer key management solutions for authentication and encryption, such as Azure Key Vault. These allow cloud services to securely store and access secrets, and automatically trigger rotation policies to retire old keys (Microsoft 2022). All of these security practices can be enforced by providing IaC modules with the desired controls built-in. This forces developers to follow secure practices, like designating network flow logs and creating secondary endpoints for key vaults, ensuring no cloud resource is created without following the security standards of the migrating company.

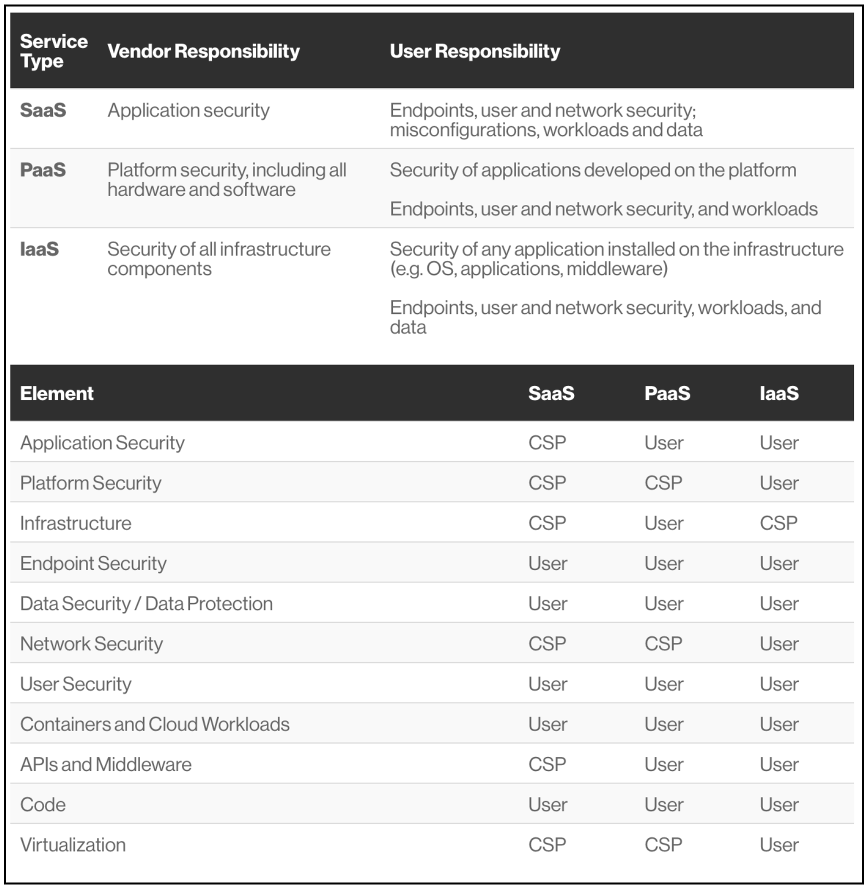
While many of these new security responsibilities belong to the migrators, many also fall on the cloud provider. This may ease the workload on companies' cybersecurity teams but adds a new level of uncertainty. After all, enterprise services are now running on computers that do not belong to the enterprise, with the cloud provider's management software holding the highest level of authority. Any vulnerabilities in this software or with its administrators pose a threat to all enterprise services involved and are entirely out of the service owner's control.

2.2.3 Industry Standards

The task of cloud migration may be complex, but its path is not entirely uncharted. Several organizations have published industry standards that give migrators a concrete set of steps to follow. One such organization, the Cloud Security Alliance or CSA, published a formidable set of around two hundred controls, which are clearly defined actions, routines, and rules to create a reliable and secure cloud environment (Cloud Security Alliance 2021). This document, known as the Cloud Controls Matrix, in its fourth version, contains 197 named controls across seventeen business categories. Each of these controls come with loose instructions to push migrators in the right direction and auditing guidelines to verify that these controls have been successfully implemented. The controls also specify the specific cloud systems they pertain to, as well as when they are applicable. For example, the control known as CEK-12, "Key Rotation", says "Rotate cryptographic keys in accordance with the calculated cryptoperiod", and indicates that it is relevant for all business domains and cloud systems (Cloud Security Alliance 2021). This control ensures that encryption keys only stay valid for a limited time and, like many other controls, makes it harder for a hacker to be granted access. Finally, the CCM contains information about which other published standards and sets of best practices contain the same guidelines, providing migrators with additional references and prompting discourse in the industry regarding any discrepancies. The CCM 4.0 is one of the most valuable resources for cloud engineers because it consolidates a tremendous amount of information, empowering diverse migration teams to divide responsibilities and confidently begin their work.

2.2.4 Cloud Service Delivery

While the Cloud Controls Matrix outlines best practices, companies must still make critical decisions related to which cloud service provider is a suitable solution for their enterprise needs. A cloud service provider or CSP is a third-party organization that leverages the pay-per-use model to sell computing resources while owning and maintaining the actual physical on-premise servers. As addressed in previous sections, this alleviates much of a customer's operational burden, enabling them to prioritize data security over infrastructure security. Together, cloud service providers and their respective client corporations comprise what is known as the shared responsibility model. The shared responsibility model is a framework that defines the split of responsibilities between CSPs and customers in a manner that promotes secure workload configuration by dictating which assets must be protected by a given party (Alvarenga 2022).



Service Type	Vendor Responsibility	User Responsibility
SaaS	Application security	Endpoints, user and network security; misconfigurations, workloads and data
PaaS	Platform security, including all hardware and software	Security of applications developed on the platform Endpoints, user and network security, and workloads
IaaS	Security of all infrastructure components	Security of any application installed on the infrastructure (e.g. OS, applications, middleware) Endpoints, user and network security, workloads, and data

Element	SaaS	PaaS	IaaS
Application Security	CSP	User	User
Platform Security	CSP	CSP	User
Infrastructure	CSP	User	CSP
Endpoint Security	User	User	User
Data Security / Data Protection	User	User	User
Network Security	CSP	CSP	User
User Security	User	User	User
Containers and Cloud Workloads	User	User	User
APIs and Middleware	CSP	User	User
Code	User	User	User
Virtualization	CSP	CSP	User

Figure 2.1 Division of Controls in the Shared Responsibility Model (Alvarenga 2022)

This arrangement is outlined in a software service level agreement or SLA published by the respective cloud provider and is one aspect organizations should carefully review when choosing a vendor for their practice. Other factors in this evaluation are related to what the provider offers in terms of reliability and performance, migration support, and data governance and security. While there are many platforms that source these on-demand cloud computing resources, Amazon Web Services, Microsoft Azure, and Google Cloud Platform are the three primary providers splitting the majority of the market share.

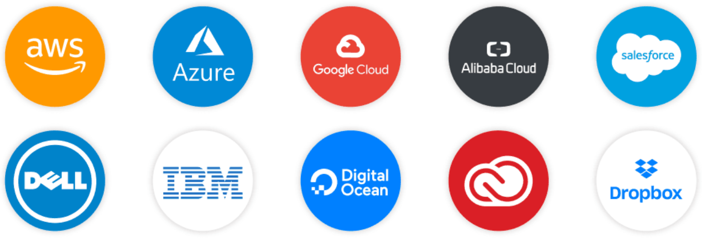


Figure 2.2 Top 10 Cloud Service Providers in 2022 (House 2021)

Offering not just industry-leading security but a substantial difference in cost effectiveness and flexibility, Microsoft Azure is the preferred choice for many enterprise organizations (Wright 2019). For a corporation like State Street that already houses the whole suite of Microsoft products, Azure was a logical vendor choice in their cloud adoption strategy. State Street manages a public cloud not only through Microsoft Azure, but also through computing resources from vendors such as Amazon Web Services and Snowflake. Throughout their migration, however, State Street has emphasized the use of cloud independent tools and infrastructure in order to promote future multi-cloud deployment options.

2.2.5 Chaos Engineering

Pioneered in 2010 by Netflix through a service known as Chaos Monkey, chaos engineering describes the process of deliberately introducing faults into a distributed system to test its resilience to random disruptions (Carey 2020). Disruptions resulting from real-world scenarios, such as natural disasters or other data/server center failures, can interrupt service to customers, leading chaos engineering to be a discipline that companies have vested interest in implementing. Emulating potential disruptions allows developers to build confidence in their software and system capabilities and offer more reliable products to their end users (Gunja 2022).

2.2.6 DevOps

Combining development and operations, DevOps refers to the automation of software delivery stages in a process that integrates development, testing, deployment, and monitoring (Ebert 2016). By automatically testing new code, deploying new production branches, and reporting problems, DevOps practices eliminate grunt-work and allow engineers to focus on concrete development.

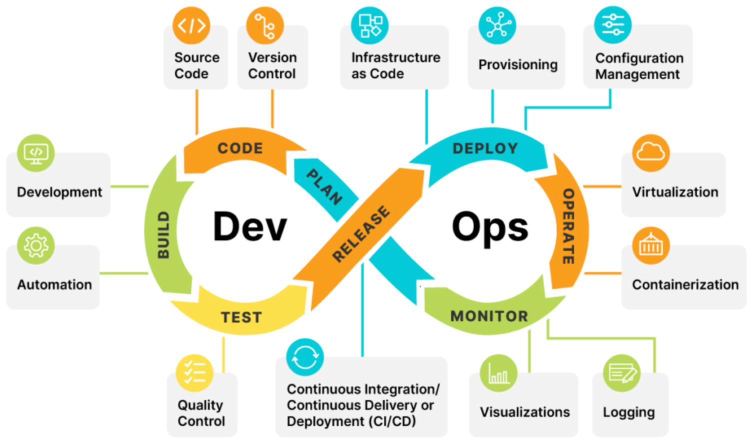


Figure 2.3 Key Development and Operations Practices (Charboneau 2022)

Implementing these practices today often involves bringing in an ecosystem of open source tools. For example, Gradle is a build tool that handles compilation scenarios such as multi-language projects. Jenkins is a popular tool for automated testing, providing a server that can be configured to automatically run tests on pushed code and report defects. Ansible is an automation platform often used for deployment, such as running new code on production web servers as soon as it gets merged to the production branch.

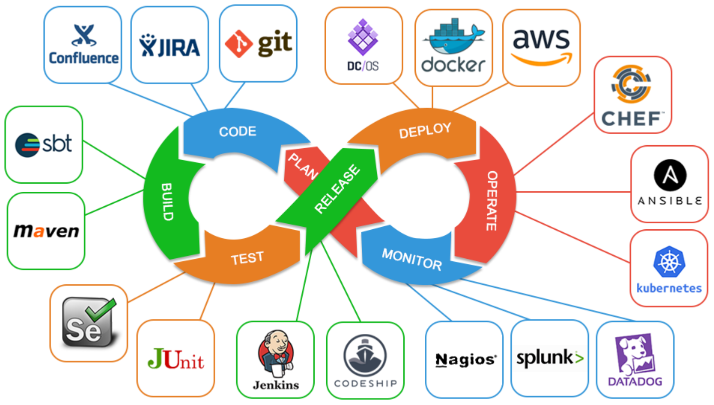


Figure 2.4 Collection of DevOps Tools (Shawn 2020)

From nightly testing to continuous-delivery, DevOps refers to a number of tools, practices, and benefits that ultimately facilitate the production of reliable software. Adopting and integrating this technical methodology into everyday workflows becomes increasingly important for companies as they work to promote strong development processes.

2.2.7 The Infrastructure as Code Paradigm

Most commonly referred to as IaC, Infrastructure as Code, as shown above in Figure 2.4, is a DevOps practice aimed towards ensuring continuous delivery. Once a lofty ideal, continuous delivery or CD has since become mandatory for organizations hoping to deliver value to their end users (Mijacobs 2022). CD is the practice where the staging environments are created by pipelines to build and deploy new changes to the codebase automatically (Mijacobs 2022). This aims to minimize the time between production releases by implementing a fail fast validation approach, allowing developers to catch minor bugs quicker and longer test cases to run only after the lightweight ones are successful. Infrastructure as Code is a way of deploying infrastructure in a way that CD pipelines maintain consistent environment settings without the need for manual configuration or reproduction (Mijacobs 2022).

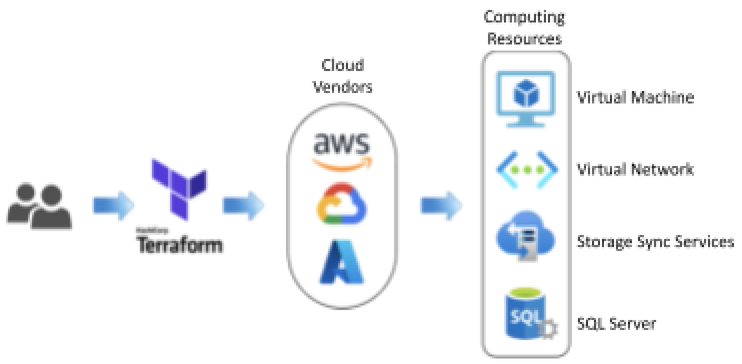


Figure 2.5 IaC Developer Workflow (LaunchRack 2022)

IaC is written in a specialized language and then interpreted by an automation tool to construct the system. These tools can either be imperative, where the steps to execute to reach the solution are defined, or declarative, where the desired state of the solution is defined and the tool chooses how to reach that (Segura 2021).

2.3 Business Risk Considerations

As a large and well-established financial institution with a large sphere of economic influence, State Street is expected to manage operational risk. In an intentional effort to preempt any such operational issues, the company structure emphasizes best practices related to legal and compliance risk management. Beyond mitigation efforts, the company as a whole has adopted a holistic view of corporate planning goals, acceptance of new technologies, and the cultivation of a culture that engages with forward-looking initiatives. In this era of rapid technological development, it is imperative for companies to expand into emerging segments in the hopes of maintaining a competitive market edge. Cloud adoption reduces costs and supports the scalability of web-based applications and database capabilities but requires a significant level of architectural and risk failure foresight to run effectively (Tafoya 2020). Our project will help State Street maintain their competitive advantage and market mobility through the automation of system resilience testing.

2.3.1 Operation Risks

Running a multinational corporation comes with a daunting amount of operational planning, and State Street's cloud migration ambitions thus warrants the consideration of risks unique to this dedication to ensure due diligence. Specifically when facilitating partnerships with third-party providers, it falls on the corporation to consider all potential operational and legal risks in the agreement. State Street utilizes some third-party applications and tools to develop and host its online platforms, however, this is not always a straightforward process and controls must be placed on which software can be utilized. For example, Apache Tomcat software, which can be used to run a Java HTTP web server on a virtual machine image, is hosted on the Azure marketplace with a generalized licensing agreement between Microsoft and Tomcat's provider. This third-party virtual machine image, however, cannot directly be used by State Street as is, as it has not been vetted for security vulnerabilities. To circumvent these issues, State Street maintains their own private image gallery to ensure their services are held up to security standards. State Street also has specialized and exclusive contracts with vendors that can specify certain indemnities and other clauses.

Investigating and moving toward new segments of technology can also require the forethought of an operational exit strategy. As State Street expands into cloud infrastructure, there is a significant amount of planning that must be dedicated to the possibility of a CSP's data retention policy expiring. The planning and mitigation behind an exit strategy consists of forming a concrete alternate architecture in the event of any data loss or other failures. State

Street satisfies this requirement through rigorous planning in the event of system crashes or overloads.

2.3.2 Change and Innovation Risks

The inherent nature of innovation is encompassed by the ideal that what is new can quickly become obsolete. As we move through this century's technological revolution, financial institutions are utilizing and investing in technology that is bound to evolve, making it imperative for State Street to adopt a culture of change that allows for constant innovation and mobility.

State Street mitigates the risk of environment transformations by rigorously scrutinizing technological adaptations to ensure a safe and secure design is in place for new software. All decisions are viewed through a holistic lens of the company’s overall goals and architecture and they employ this system of checks and balances to verify that no drastic changes can be made without explicit approval. Adopting new technology can come with risks, and as organizational change is implemented, there remains a need for an overarching emphasis on forward-thinking infrastructure during the development process. Encouraging an innovative mindset for the application owners and developers encourages productivity, supporting acceptance for new initiatives across the entire company.

2.3.3 Financial and Reputational Risks

As a major player in the finance industry, State Street must investigate possible integration opportunities for new technologies within existing applications to remain a leading institution.

While our team developed computing resources using the Microsoft Marketplace, we were cautioned against utilizing some of the software that requires paid credits. In 2021, State Street was estimated to spend $622.7 million on Information and Communications Technology, and although this figure accounts for extraneous costs, the company still must carefully track its unplanned expenses to comply with its strategies for digital transformation (Globaldata 2022). These initiatives involving expansion of technology logically demand investment to support new software licenses and resources, however, the costs can quickly surpass initial budgets if no overarching corporate direction related to these advancements is defined and adhered to.

Beyond digital preparedness, State Street relies on a positive reputation to promote trust among its customers and investors. At the highest level, our project is focused on the transference and protection of sensitive information. Working to host large amounts of protected client data in the cloud requires a generous amount of forethought and planning to ensure all information is handled securely. Generating a sense of stability surrounding the financial sector is imperative to maintaining a level of reliability and stakeholder confidence. State Street reinforces its reputation through public and third-party auditing of its internal systems and conducts rigorous penetration testing on a yearly basis. By outsourcing an annual stability assessment, State Street is able to conduct unbiased appraisals of its internal frameworks and earn client confidence by encouraging transparency about the security their systems provide.

2.3.4 Training Risks

Coupled with the evolution of advanced technologies, there is a parallel risk in the market need for technical experts, specifically in the field of cloud development. The whirlwind of technological evolution in the last decade has created a dramatic skills gap, where employees lack the skills to keep pace with their adapting job descriptions (Trevino 2022). As cloud engineering has matured, corporations are looking to educate current employees and hire a new workforce to meet this competency gap. With proficient knowledge and capabilities of Terraform and Azure difficult to find among potential recruits, it has proven challenging to invest in cloud development talent, as companies need experts and dedicated workers to maintain large corporate systems. Companies not only need to prioritize recruiting and retaining top talent, but also reskilling current employees to keep them competitive and successful (Trevino 2022). While internal learning programs can educate team members, it is often still easier and more seamless, though sometimes pricier, to contract individuals who already possess expert knowledge. Outsourcing risks corporate stability and resources but can drastically improve productivity when migrating to new technologies. This new workforce prompts risks related to onboarding and training activities and grants access to privileged information to external parties. In the event of intrusive attacks or malicious actions by internal parties, there are drastic potential ramifications to State Street's internal architecture, including the delay in fixing the affected systems. For this and other reasons, it is business critical for State Street to conduct thorough risk assessments and secure top legal assets to manage and supervise their affairs.

3. Software Development Methodology

Within a professional organization, administrative management is a critical aspect of a project's successful development and execution. Managers not only look to direct their team towards achieving strategic goals but also to optimize resource use and decrease involved risk in any given undertaking. In order to coordinate all of these diverse activities while remaining cognizant and prioritizing scope, budget, and stakeholders, many organizations choose to leverage a project management methodology to structure their workflow. The general objective of a project management methodology is to aid in the tracking and governance of a project, allowing team leads to effectively facilitate team collaboration and maintain a holistic view of the project. This set of principles, techniques, and best practices, is usually specialized to a group but standardized across an organization and creates a structured atmosphere that promotes productivity and is adaptable to new challenges. Various industries or project teams will utilize different methodologies according to their specific needs and environments; the technology industry, especially financial corporations, often utilizes the agile scrum methodology, which emphasizes incremental and flexible development.

3.1 Agile Scrum Methodology

The agile scrum methodology is a project management system that weds the philosophy of agile development with the framework of scrum. The principles of agile project management emphasize customer satisfaction and seek early and continuous delivery (Beck et al. 2001). The agile philosophy stresses building development critical features first to create a minimally viable, deliverable product.

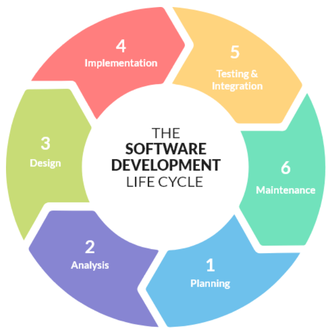


Figure 3.1 Agile Methodology (Kumar 2021)

The non-essential features are built in subsequent stages of development, with the timeline and priorities adjusted based on client feedback. One of the key aspects of agile is its iterative nature; planning and goal setting occur frequently to ensure the team has focused objectives and the product is incrementally delivered to stakeholders, providing them with the highest business value in the shortest turnaround time. The continuous involvement of clients enables organizations to readily pivot project objectives to adapt to changing business needs.

As a whole, the scrum methodology promotes cohesion among teams working toward a primary objective, however within this cross-functional unit of professionals, there are three key roles within scrum that help distribute product-related responsibilities and maintain accountability (Schwaber 2020). The product owner represents the software users and is responsible for maximizing the value and return on investment of the undergoing development, driving the product from a business perspective. The scrum master is responsible for the overall effectiveness of the team, helping to remove hindrances to the project's progress and ensure a smooth and consistent adoption of Scrum principles into the team's workflow. The third and final key role in the scrum methodology is the team of developers, whose work is broader in domain and directed toward creating a usable deliverable.

Scrum project management is a form of scaled agile development that emphasizes self-organization and is characterized by short phases of project work known as 'sprints,' which are typically structured as two-week cycles. Within each sprint, there are many events that help facilitate the nature of development, creating regularity and maximizing efficiency by strictly designating and structuring meetings. There are four of these events or ceremonies, sprint planning, daily scrum, sprint review, and sprint retrospective, which all aid in effective expectation setting and conversation facilitating. During the sprint planning period, the team works to identify a selection of the overall project scope to work on in the upcoming cycle. This meeting is designated to occur at the onset of each new sprint and involves all members of the scrum team. The ultimate outcome of sprint planning is the creation of a sprint backlog, also known as the sprint goal. The sprint backlog represents a shippable series of tasks, where this unit of work is an increment taken from the product backlog that the team is committed to for the given sprint. In preparation for this ceremony, the product owner has the responsibility of composing the product backlog, which entails detailing acceptance criteria and other requirements to help the team properly forecast the expected timeline of various tasks and assess what amount of work can feasibly be accomplished. The duration of sprint planning is directly correlated to the duration of the sprint and lasts a maximum number of hours equal to double the number of weeks in the sprint.

Daily scrum meetings, sometimes referred to in industry as 'Standups', are briefer meetings where the development team synchronizes their activities for the day, sharing what they intend to accomplish and any obstacles they may need assistance with. These roadblocks are undertaken by the scrum master while the development team communicates their progress in the past day and identifies opportunities to sync up with one another after to engage in longer, more technical conversations or collaborative work. The next two ceremonies occur once a sprint is completed. The sprint review is a scrum event attended by all members of the team as well as external clients and is a meeting where working software is demonstrated in a forum that allows stakeholders to give actionable feedback. The progress made on the deliverable is assessed and user story completion is also evaluated against the defined acceptance criteria. These comments are then adapted into items, entered into the product backlog, and assigned precedences. Finally, the sprint retrospectives that follow the conclusion of each sprint are a forum for teams to reflect on what was accomplished in the sprint and identify any areas for improvement for the next cycle.

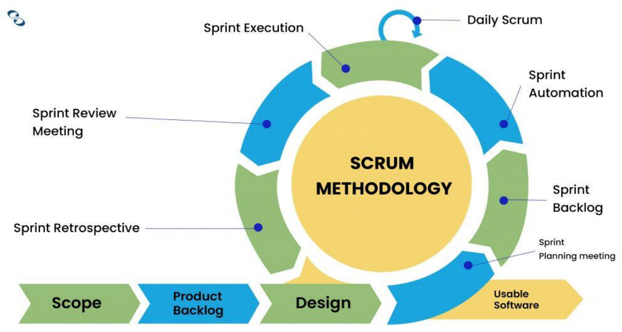


Figure 3.2 Scrum Framework (Casey 2022)

The last core component of the scrum development framework is the scrum artifacts, which are used to represent the work done or value added in a way that promotes transparency among the team and clients. The product backlog is an ordered list managed by the product owner where each item represents something needed to improve the product. Each of these broader elements is converted into an increment of value for a team member to work on during a sprint, known as a user story. A user story conveys functionality requirements defined by the user to achieve some given business value. Each story can be validated through acceptance tests and must meet the team-defined criteria for 'done' in order to be accepted by the product owner and incorporated into future releases. Attached to each user story are 'story points' that are used to estimate the value of the respective feature in terms of how involved the development will be, which also gives insights into the duration of the given feature(s).

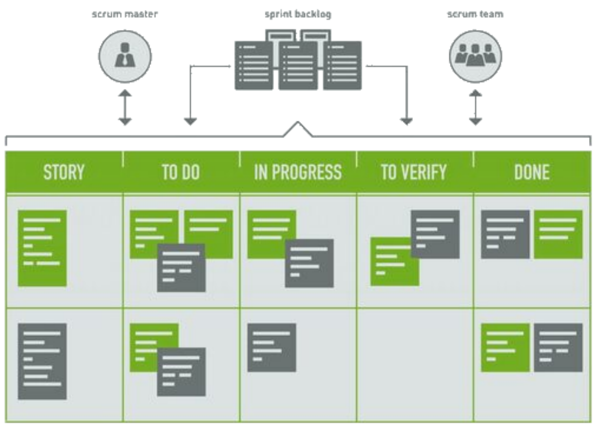


Figure 3.3 Scrum Task Board (Digite 2019)

While agile and scrum differ slightly in the flexibility offered, the systems complement each other well when implemented jointly in the agile scrum project management framework.

Outside of the team accountability and progress transparency that agile scrum helps prioritize, the framework is easily scalable and adaptive and overall achieves better and more regular deliverables. The agile scrum methodology is used by companies of all sizes for collaborative project efforts, and its sprint-based nature enables fast and flexible project work, making it a popular choice, especially for teams working in software development (Digite 2019).

3.2 Alternative Project Management Methodology

Beyond the agile scrum methodology, there are several other management philosophies that are frequently employed by teams and certain industries when a given style is more suited to the competencies, expectations, and flexibility of the operation at hand. While some methodologies prioritize continuous improvement, minimizing inefficiencies, or dependency networking, the most common approach to project management other than agile is a very traditional one known as the waterfall methodology.

Waterfall methodology emphasizes linear or rather sequential development; it is a simple and rigid architecture that is ideal for projects with one final substantial deliverable, for example, construction jobs. Overall, this form of management maintains a high degree of control over subtasks and follows a strict project schedule, with each phase cascading seamlessly into the next, hence the name. In the technology industry, the phases of waterfall methodology correspond to the phases of the software development lifecycle, which is a conceptual framework used to see an application through completion. Each of the consecutive phases defines a concrete milestone that must be fully executed before the team traverses to the next.

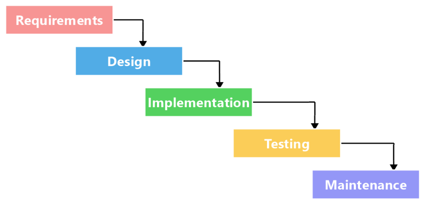


Figure 3.4 Waterfall Methodology (Finlay 2022)

The first stage of the progression is requirement gathering. Within this planning period, the team frequently meets with stakeholders to assess project requirements, establish thorough documentation, and collect other information necessary for later stages of development. Common discussions that take place related to the project revolve around business risks, success metrics, costs, assumptions, and other dependencies and limitations (Abode Communications Team 2022).

The outlined project requirements are then used as a framework in the design phase, where an acceptable technical solution is produced. A high-level design is conceptualized to address the business incentive and project scope using workflow diagrams, layout mockups, architecture documentation, and data models (Adobe Communications Team 2022). This logical overview is then converted into a concrete design that outlines the user interface and experience, specific hardware and software technologies, and integration details (Adobe Communications Team 2022). Stakeholder input is an important aspect of these determinations as far as resource and tool needs are concerned, as well as discussions about potentially outsourcing work to contractors.

Following a comprehensive design period, the technical implementation of this plan begins. This is typically the shortest phase where working environments are set up and development occurs. Oftentimes, once the majority of the code is written, integration of the software with other services and applications will occur. This can be a very involved process requiring a significant amount of collaboration, depending on which business processes or third-party providers need to be integrated with.

Throughout the development and integration process, the team will be continuously testing it for stability and defects; however, following the conclusion of code development, a proper acceptance or final testing stage occurs. This period of quality assurance checks the application against the established requirements and ensures that the client is completely satisfied with the end product. Alongside this, the team verifies the functionality of the application, thoroughly testing for bugs or issues and validating other acceptance criteria.

Once the software is deployed or made available on the market, the maintenance phase commences. Any issues that arise or are discovered by the client are systematically patched by the team, and overarching strategies are devised for the purposes of updating and upgrading the application as needed. Continuous support is provided by the team towards this service beyond the project life cycle for the duration of the application lifetime, and as a result, this phase can often be very long due to the extensive involvement even minor bug fixes require.

While agile scrum methodology touts iterative and changeable product management, waterfall development has the advantage of an explicitly detailed timeline of product delivery. This, alongside the ability for more accurate budgeting and cost reporting, helps establish and maintain a strong and dependable business reputation. Waterfall methodology may initially seem like a strong choice from the perspective of reliable customer delivery; however, it places a lot of upfront responsibility during the requirements gathering stage. The client must communicate all of their prerequisites early in the development process, as this style of management does not offer the opportunity for ongoing stakeholder feedback. Following the software development life cycle thus oftentimes results in a project that is very difficult to pivot and can readily lead to excessive delays. With only one term to produce a minimally viable product, we favored more flexibility and decided to utilize the agile scrum methodology.

4. Software Development Environment

With State Street looking to transition primarily to cloud-hosted and cloud-native applications in the next couple calendar years, our project architecture and technology stack was very structured and specific as we worked with the company to develop long-term plans to ensure resiliency. The software industry has provided many tools to aid companies in this process as they migrate their services to these digital cloud environments. These include tools like Azure DevTest Labs that aid in the rapid development and functional testing of services, tools like Terraform that define and deploy specialized cloud environments, and tools like Azure Chaos Studio that stress test deployed services against disaster scenarios. These platforms, provided to us alongside State Street’s project management software, allowed us to execute artificial failures on their systems while developing in a manner that would facilitate a multi-cloud adoption strategy in the future.

4.1 Project Management Software

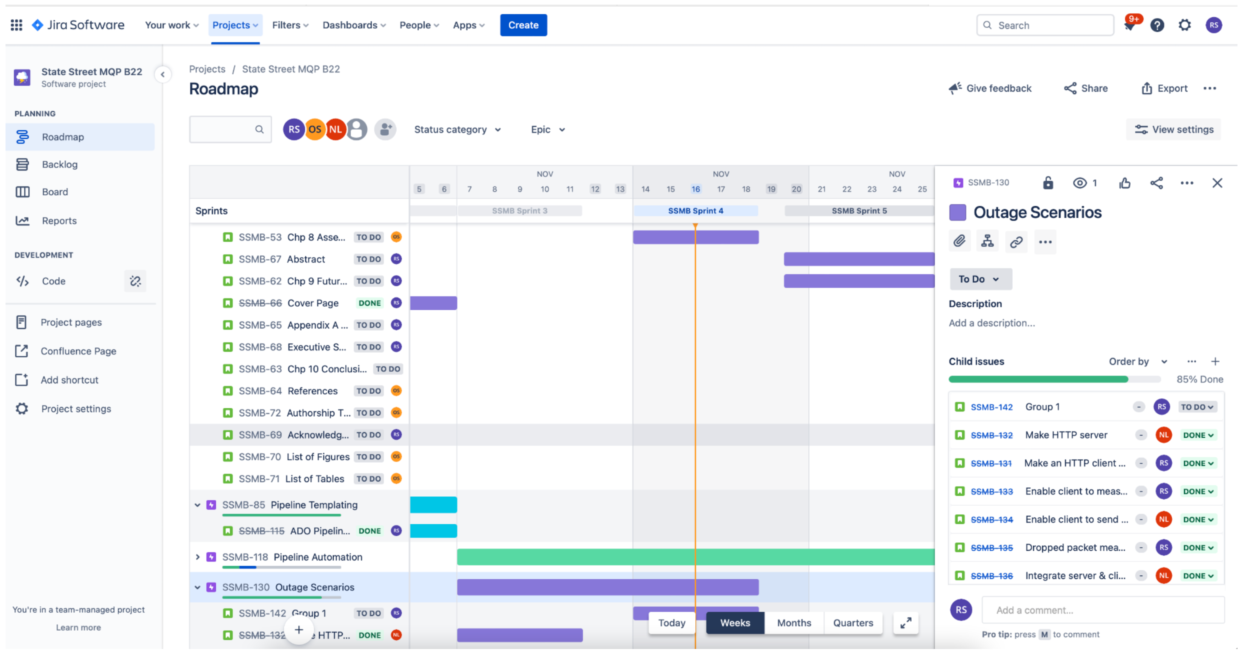
One of the main challenges that project managers face when overseeing a large development problem is that of assessing, addressing, and allocating the workload. As companies and teams undertake increasingly involved projects, it becomes critical to rely on project management software and tools that are digital platforms specially designed to assist with project scheduling and resource allotment. By utilizing these collaborative technologies, much of a manager’s organizational responsibilities are relieved, enabling them to devote their efforts towards development or other administrative priorities.

4.1.1 Confluence

For all of our documentation and learnings, our team used Confluence. Confluence, a cloud site hosted by Atlassian where updates are regularly released and automatically integrated, is a popular choice among corporations for such documentation technology (Atlassian 2022b). State Street is one such company that leverages the services of Confluence to act as a repository for project information. Alongside accessing company materials, architecture diagrams, and other resources through State Street’s enterprise subscription, we also used the free billing tier externally to manage our own team’s documents. Throughout development, Confluence assisted us with knowledge sharing, understanding existing infrastructure, strategic planning, and creating more actionable project descriptors.

4.1.2 Jira

To keep track of the project responsibilities, our team used Jira. Jira is another subsidiary of Atlassian that serves as an issue-tracking platform and can be seamlessly integrated with Confluence by allowing user stories to be linked to specific project requirements and other documentation sections (Atlassian 2022a). Jira Software is an agile management tool with a very comprehensive set of features that eases sprint planning, backlog grooming, story point tracking, and version management (Atlassian 2022a). With State Street's enterprise license, we established an epic, or a collection of user stories, related to Chaos Engineering and ensured best practices for progress reporting for future tracking. Similarly to Confluence, we also created an external Jira Board with a copy of this epic and an epic for paper writing.



Projects / State Street MQP B22

Roadmap

Sprints

SSMB-53	Chp 8 Asse...	TO DO	OS
SSMB-67	Abstract	TO DO	RS
SSMB-62	Chp 9 Futur...	TO DO	RS
~~SSMB-66~~	Cover Page	DONE	RS
SSMB-65	Appendix A ...	TO DO	RS
SSMB-68	Executive S...	TO DO	RS
SSMB-63	Chp 10 Conclusi...	TO DO	
SSMB-64	References	TO DO	OS
SSMB-72	Authorship T...	TO DO	OS
SSMB-69	Acknowledg...	TO DO	RS
SSMB-70	List of Figures	TO DO	OS
SSMB-71	List of Tables	TO DO	OS
SSMB-85	Pipeline Templating		
~~SSMB-115~~	ADO Pipelin...	DONE	RS
SSMB-118	Pipeline Automation		
SSMB-130	Outage Scenarios		
SSMB-142	Group 1	TO DO	RS
~~SSMB-132~~	HTTP...	DONE	NL

SSMB-130

Outage Scenarios

To Do

Description

Add a description...

Child issues — Order by — 85% Done

SSMB-142	Group 1	RS	TO DO
~~SSMB-132~~	Make HTTP server	NL	DONE
~~SSMB-131~~	Make an HTTP client ...	RS	DONE
~~SSMB-133~~	Enable client to meas...	RS	DONE
~~SSMB-134~~	Enable client to send ...	NL	DONE
~~SSMB-135~~	Dropped packet mea...	RS	DONE
~~SSMB-136~~	Integrate server & cli...	NL	DONE

Add a comment...

Pro tip: press M to comment

Figure 4.1 Jira Board User Story Statuses

4.2 Integrated Development Environment

In computer programming, an integrated development environment is an application that consolidates the tools needed for developing software, providing engineers with a text editor, debugger, and compiler all encapsulated into a user-friendly interface (Codecademy 2022). While most of our infrastructure was managed in external environments, our work in modifying configuration files and infrastructure specifications during initial development was facilitated by the use of an IDE.

4.2.1 Visual Studio Code 1.70.1 and 1.72.2, Git 2.37.2.2

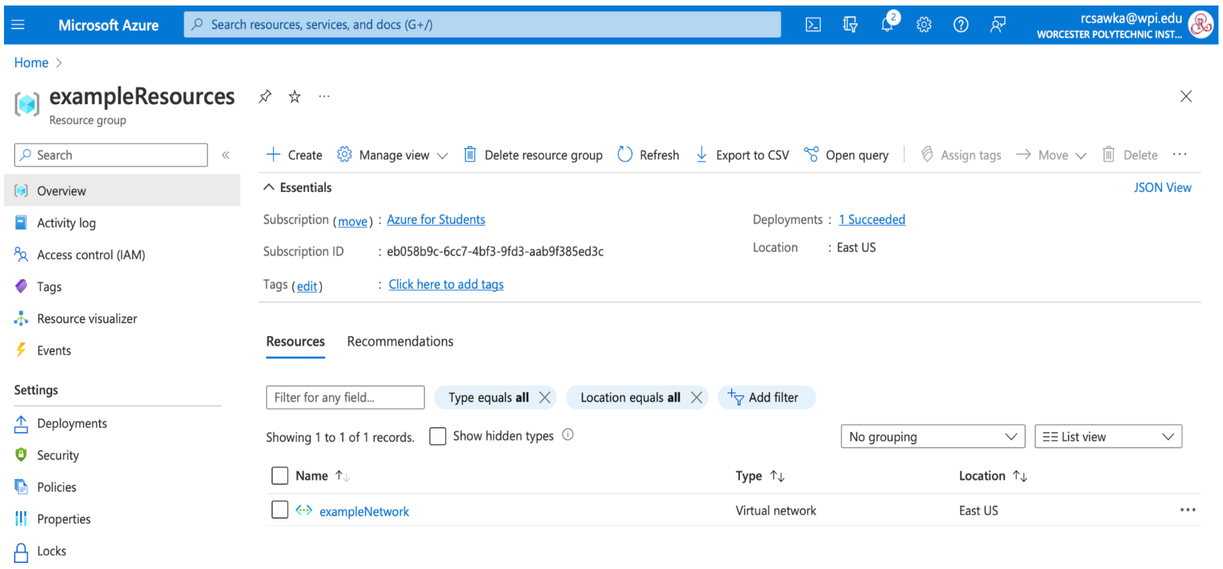
Visual Studio Code is an open-source product developed by Microsoft that we selected as our source code editor (Microsoft 2021). VS Code facilitated our work in building the pipeline and developing client/server applications, and we utilized git to manage version control and branching within our repository. The Azure DevOps pipeline that our team constructed was stored in a git repository, requiring us to interface with git in order to make modifications. We utilized Git Bash as our shell program in order to run these update commands as the PowerShell extension typically integrated into Visual Studio was disabled by our system administrators to follow the Center for Internet Security's or CIS security benchmarks, preventing us from defaulting to the embedded git feature in our IDE.

4.3 Additional Software Tools

Over time, the process of provisioning and managing infrastructure has been codified and more frequently handled through machine-readable definition files as opposed to the previously manual processes (Merron 2021). This IaC automation technology is a solution that works for both cloud environments and on-premises computing resources. These IaC tools were the primary technologies that we interacted with on this project in order to follow the company-outlined best practices aligned to our scope in both the writing of pipeline configuration modules and the deployment of fault injectable resources.

4.3.1 Azure Resource Manager

Within the broader cloud computing and application hosting platform known as Microsoft Azure is a service known as the Azure Resource Manager or ARM. ARM is what we used to create, manage, and delete cloud resources (Fitzmacken 2022).



exampleResources
Resource group

Essentials

Subscription (move) : Azure for Students
Subscription ID : eb058b9c-6cc7-4bf3-9fd3-aab9f385ed3c
Tags (edit) : Click here to add tags
Deployments : 1 Succeeded
Location : East US

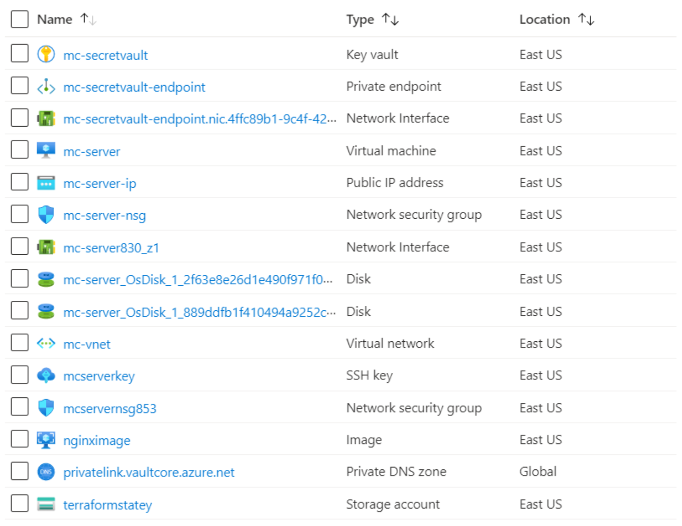
Type equals all
Location equals all

Showing 1 to 1 of 1 records. Show hidden types

Name	Type	Location
exampleNetwork	Virtual network	East US

Figure 4.2 Azure Resource Manager's Centralized Dashboard

The centralized dashboard eased our development as we sent requests through the portal to provision and deploy our various desired resources, such as virtual machines, virtual networks, key vaults, and others. We used Azure Resource Groups as a means of containerizing our resources into collections of assets. We also leveraged Terraform, described in section 4.3.2, to enable independence from a single cloud provider.

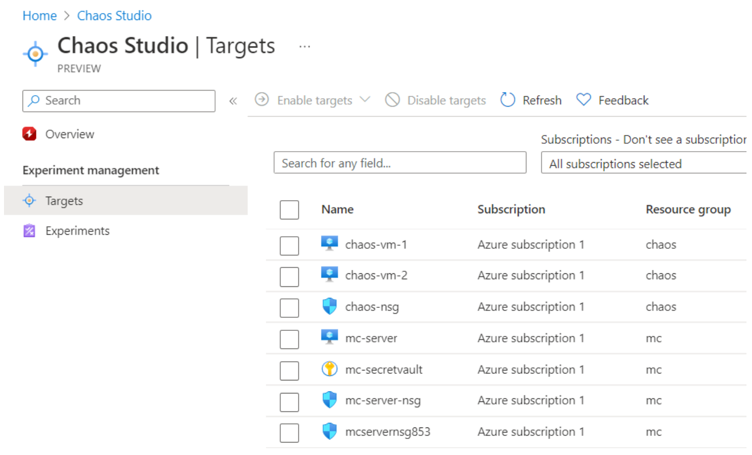


Name	Type	Location
mc-secretvault	Key vault	East US
mc-secretvault-endpoint	Private endpoint	East US
mc-secretvault-endpoint.nic.4ffc89b1-9c4f-42...	Network Interface	East US
mc-server	Virtual machine	East US
mc-server-ip	Public IP address	East US
mc-server-nsg	Network security group	East US
mc-server830_z1	Network Interface	East US
mc-server_OsDisk_1_2f63e8e26d1e490f971f0...	Disk	East US
mc-server_OsDisk_1_889ddfb1f410494a9252c...	Disk	East US
mc-vnet	Virtual network	East US
mcserverkey	SSH key	East US
mcservernsg853	Network security group	East US
nginximage	Image	East US
privatelink.vaultcore.azure.net	Private DNS zone	Global
terraformstatey	Storage account	East US

Figure 4.3 Resources Housed in an Azure Resource Group

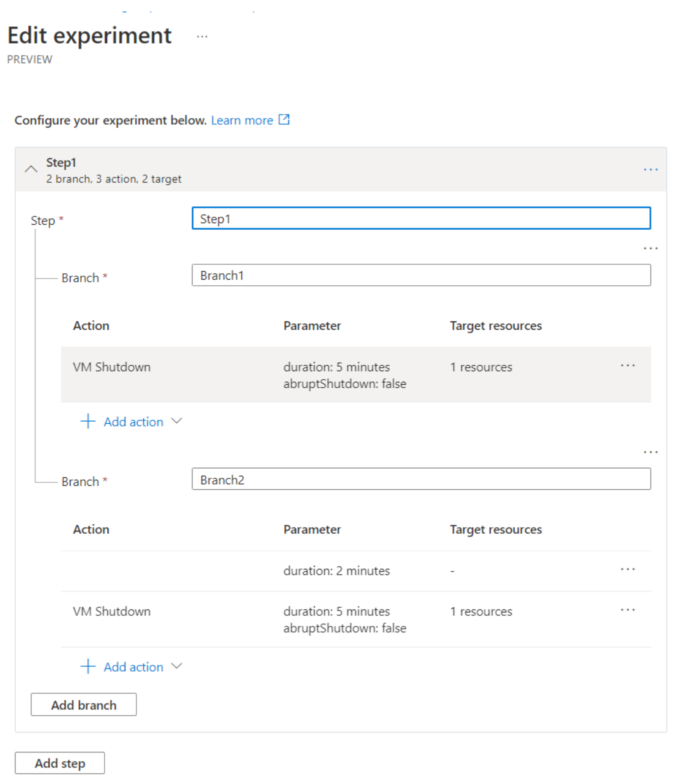
4.3.2 Azure Chaos Studio

Azure Chaos Studio is a tool that allows developers to simulate a number of different resource failures, including data center outages and network segmentation (Isaiah 2021). Even with protections in place, cloud providers' data centers can go offline no matter how many scenarios the enterprise developers test. Since enterprise service providers cannot guarantee 100 percent reliability of all their cloud resources, they must be ready to deal with outages. Azure Chaos Studio enables developers to see how their cloud environments behave during these unfortunate events, measure relevant information, and track improvements as they use this data to harden their systems. With Azure Chaos Studio, cloud developers can minimize damage and maximize recovery speed, ensuring that their environments are prepared for the worst (Isaiah 2021). We used Azure Chaos Studio to deliberately simulate failures by enabling our resources as targets for experiment fault injections.



Name	Subscription	Resource group
chaos-vm-1	Azure subscription 1	chaos
chaos-vm-2	Azure subscription 1	chaos
chaos-nsg	Azure subscription 1	chaos
mc-server	Azure subscription 1	mc
mc-secretvault	Azure subscription 1	mc
mc-server-nsg	Azure subscription 1	mc
mcservernsg853	Azure subscription 1	mc

Figure 4.4 Agent-Based and Service-Direct Fault Target Resources



Edit experiment ...

PREVIEW

Configure your experiment below. Learn more

Step1

2 branch, 3 action, 2 target

Step * Step1

Branch * Branch1

Action	Parameter	Target resources
VM Shutdown	duration: 5 minutes abruptShutdown: false	1 resources

Add action

Branch * Branch2

Action	Parameter	Target resources
	duration: 2 minutes	-
VM Shutdown	duration: 5 minutes abruptShutdown: false	1 resources

Add action

Add branch

Add step

Figure 4.5 Azure Chaos Studio Experiment Construction Page

4.3.3 Azure Command Line Interface

Azure CLI provides a command line interface to communicate with the Azure Resource Manager in a concise manner, allowing reliable control and providing automation opportunities. Like any reputable command line interface, Azure CLI comes with clear command definitions, unaffected by UI changes or browser incompatibility. It enables concise documentation of complex processes, aiding new users learning Azure, and making it easy for experienced users to share their knowledge. Finally, a developer can integrate Azure CLI commands with the tools of modern scripting languages, and automate complex tasks with ease.

4.3.4 Terraform

Terraform is a tool that handles the provisioning of cloud environments from IaC. Cloud environments are complicated, often requiring specialized virtual network definitions, multiple virtual machines and an impervious set of security rules, and cannot be easily replicated manually. To automate this replication, developers utilize IAC tools like Terraform. By providing software to interpret textual definitions of cloud environments, IaC tools allow developers to quickly provision entire environments with one human readable file. This provisioning can be invoked from a command line interface, so developers do not need to click through any web GUIs. This cloud defining code can be modularized and reused, allowing developers to fine tune cloud environments like they fine tune the code that runs in them. Another benefit of Terraform is that much of its syntax is independent of any single cloud provider, making it relatively easy to migrate from one provider to another, or utilize multiple providers at the same time. While Terraform provides the necessary IAC capabilities on its own, we also utilized Terragrunt, a thin wrapper for Terraform that eliminates a need for duplicated code by generating IAC from Terraform modules. During the development of our framework, we modified Terraform modules released by the Azure team, and wrote Terragrunt files that used the modules to provision entire environments with a single command.

4.3.5 Kubernetes

Kubernetes is a container orchestration tool that manages the deployment of container jobs to available compute resources, and provides a communication substrate to network these containers together (Kubernetes 2022). Kubernetes can abstract a complicated distributed computer of many physical and virtual machines into a single machine that runs and provides an environment for containers. A kubernetes cluster receives a deployment, sometimes involving many containers and network configurations, and automatically starts containers on available physical computers, monitoring their status and fostering communication between them. Being an excellent tool for running microservices, kubernetes is an essential asset for developers of modern distributed applications.

4.3.6 Zoom 5.12.3 (9638), WebEx 42.4.0.21893

Zoom and WebEx are cloud-based video conferencing services that we used throughout the duration of our project to conduct meetings with our team, sponsors, and advisors. Our team often connected in person in a conference tech suite, however, we primarily hosted our daily standups, sprint planning sessions, and sprint retrospectives virtually over Zoom. Our weekly check-in meeting with our WPI advisors and our daily check-in meetings with our State Street sponsors were also held over Zoom. We frequently leveraged the recording functionality provided by Zoom to document our meetings so that we could later reference information provided to us by our sponsors. In these meetings, as well as the broader State Street standups held using the WebEx platform, screen sharing was also a very useful tool and made remote collaboration significantly more convenient.

4.3.7 Microsoft Teams 1.5.00.28361, Microsoft Outlook 2202, iOS Messaging

For asynchronous communication with our sponsors and our advisors, we used our WPI education and State Street enterprise Microsoft Outlook accounts. For internal State Street communication with team members and other support beyond our primary sponsors, we used Microsoft Teams for instant messaging purposes. Finally, within our immediate project team, we used iOS Messaging as a means of casual real-time communication.

5. Software Requirements

5.1 Software Requirements Gathering Strategy

Prior to and following the onset of our employment period, we worked alongside our sponsors to define the project requirements and determine the trajectory of the company's cloud engineering strategy proposal. The duration of our software requirements gathering process, however, was essentially minimal as it was largely outsourced as our sponsors' responsibilities in advance of our onboarding. Our daily meetings helped narrow down the scope of our project to what could feasibly be accomplished during our time with the company and identify the path of development that would provide the greatest business and educational value. At both the end of our first full week and the beginning of our second, our scope and requirements were still evolving even as we carried out the initially designated tasks. Toward the end of the third week we also experienced another change in direction as to how we decided to approach our minimum viable product as a result of unforeseen issue blockers. As we proceeded through development, our sponsors monitored our progress and we sought out regular approval on the advancement of our user stories.

5.2 Functional and Non-Functional Requirements

The chaos engineering framework establishes a process for simulating disaster scenarios with controlled experiments and uses the resulting data to strengthen the systems involved. The framework is a computer program that automates chaos experimentation by provisioning environments, preparing an experiment, running the experiment, outputting data, and optionally destroying any cloud resources involved. Along with this program, we provide a set of documentation that explains the framework, and provides relevant examples for how to use it. These examples are complete with IaC, CLI commands, and other files as needed.

The framework documentation includes examples for the following scenarios:

- Unplanned VM shutdown
- CPU/Memory overload
- Access policy violation
- Kubernetes pod failure

In each example scenario, the documentation explains the causes and effects of the system failures being simulated in a way that enables developers to identify additional environments vulnerable to the same scenario. The documentation provides concise steps to conduct these experiments manually, and explains exactly how the framework automates each step. All IaC is provided for each example, which includes an explanation of any required inputs that need to be provided to the IaC in order to provision the resources to achieve the desired environment. After providing the steps to conduct an example experiment manually, the documentation explains how to automate the experiment with our framework by invoking the scripts directly, through the docker container, and through an Azure DevOps pipeline.

5.3 User Stories and Epics

Sprint	User Story	Points
Epic: Chaos Engineering Background		
0	Develop project scope	4
0	Manually run VM outage chaos experiment	4
Epic: Pipeline Templating		
1	Debug permission errors in creating cloud resources in Azure portal	8
1	Follow IaC developer workflow to avoid authorization blocks	10
1	Identify virtual machine image IaC	2
1	Procure correct VM image developer workflow to avoid authorization blocks	7
1	Procure working virtual network and subnet	3
1	Configure remote Terragrunt state	3
1	Naming modifications to plateng pipeline	4
1	Debug missing PubSub provider	2
1	Add AzureRM provider with PubSub alias to Terragrunt configuration	1
2	Configure correct DNS zone for key vault's private endpoints	2
2	Identify primary and secondary spoke virtual networks	1
2	Create primary and secondary subnets	1
2	Identify primary and secondary hub virtual networks	1
2	Identify DNS zones inside hub networks	1
2	Resolve key vault creation conflict in application pipeline	3
2	Identify Terraform pipeline's missing key vault service principle from the logs	1
2	Debug key rotation policy error on applying new DES key	2

Epic: Experiment Data Collection Tools		
2	HTTP Request Server Development	0.5
2	HTTP Request Client Development	0.5
2	Server and Client Integration, Data Collection	3
Epic: Chaos Experiments		
3	Resolved Chaos Studio subscription registration	0.5
3	Wrote initial deliverable framework	1
3	Conducted and documented Group 1 experiments	3
3	Automation scripting	6
3	Conducted and documented Group 2 experiments	2
4	MVP deliverable draft	3
4	Conducted and documented Group 4 experiments	5
4	Conducted and documented Group 3 experiments	10
4	Kubernetes IaC	3
4	Kubernetes CLI scripting	4
4	Kubernetes pod fault experiment automation	5
4	Generalized script support for service/agent faults	5

Table 5.1 Project Epics and User Stories

6. Design

Our final deliverables were:

1. Python scripts to prepare an Azure Chaos experiment from a JSON file with the experiment configurations
2. Bash scripts to prepare an environment and run multiple experiments from IaC and JSON
3. Text file documentation with the four working example scenarios including the IaC, experiment JSON, and pipeline YAML files

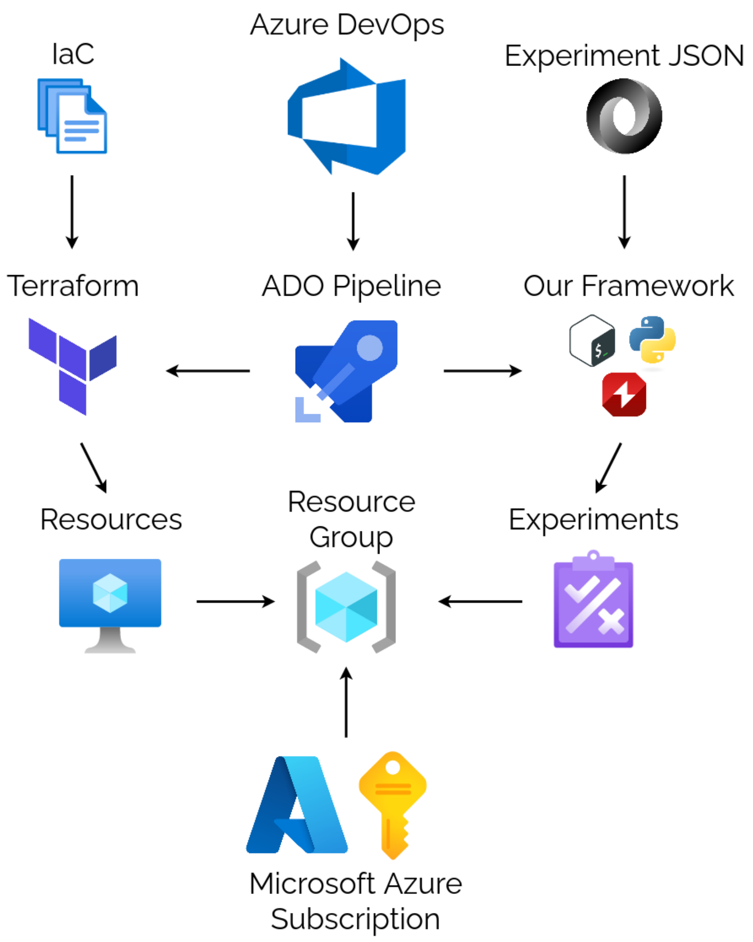


Figure 6.1 High-Level Project Architecture

Figure 6.1 above presents a pictorial overview of the systems and software that our team used to deliver our objectives. At the highest level of our project flowchart is Microsoft Azure, specifically the DevOps and Chaos Studio services. Using both the State Street Azure subscription as well as our own personal student subscriptions we were able to create, manage, and target resource groups in our experiments.

6.1 Pre-Existing Architecture

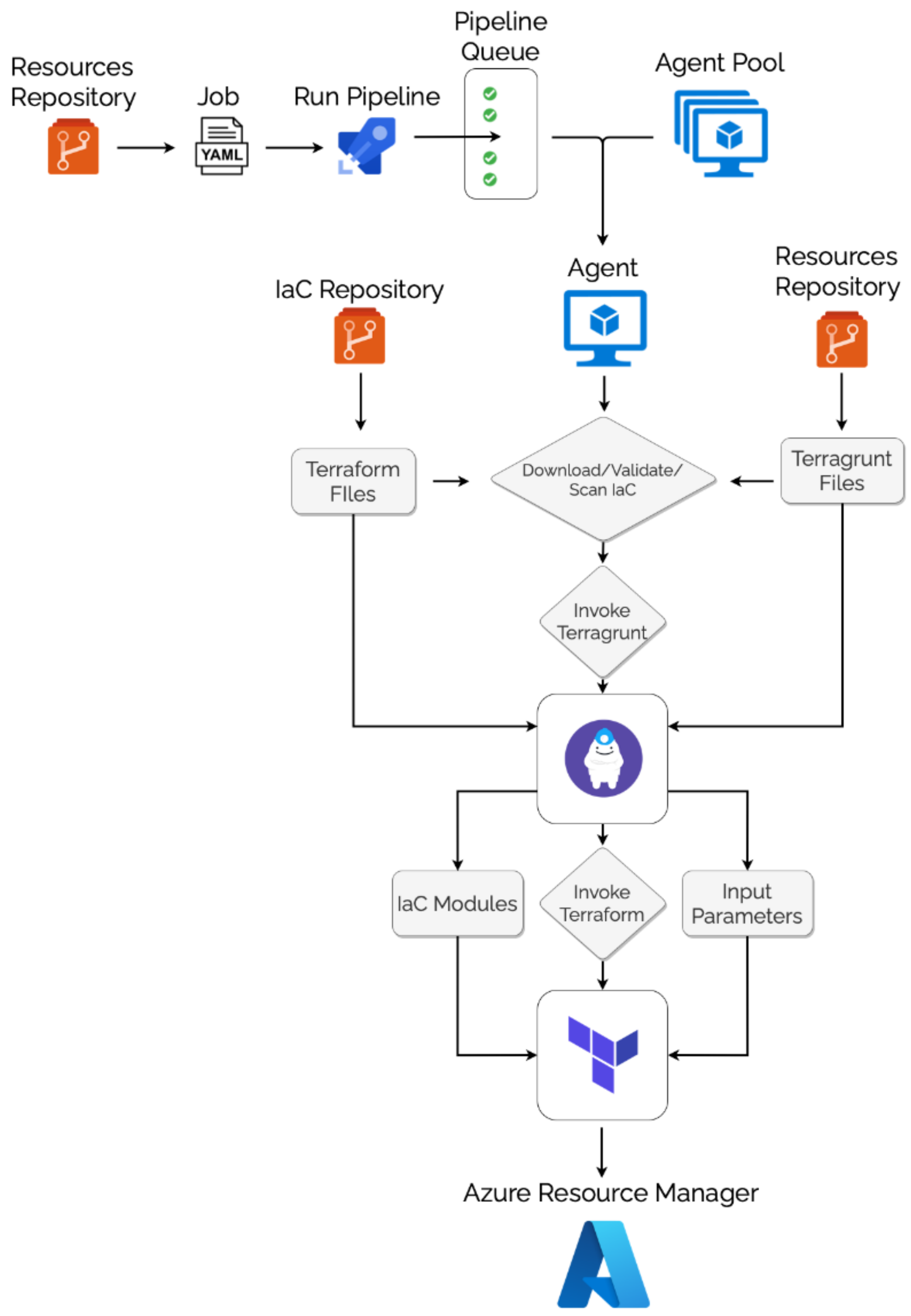


Figure 6.2 Cloud Infrastructure Provisioning Pipeline

Figure 6.2 shows a process that allows cloud engineers to automate the creation of cloud environments by utilizing Azure DevOps pipelines. In this process, the installation and configuration of IaC tools is automated along with their use, allowing not only the resulting cloud environments to be clearly defined, but also the environments that provision them. This permits developers to easily limit access to their cloud resources, clearly define the right IaC repositories, and ensure proper use of the provisioning tools. The nature of automating pipelines with Azure DevOps also enables developers to start provisioning jobs on dedicated machines rather than their own workstations, allowing them to use their workstation freely for other tasks while still being able to monitor the status of the automated process.

6.2 Our Framework

Our framework is a combination of bash and python scripts that invoke Terraform and Azure CLI. These scripts are packaged in a docker image, designed to be stored in a container registry, pulled and invoked by a DevOps pipeline, and provided with inputs through environment variables. The container optionally pulls and provisions IaC from a git repository, and then conducts chaos experiments.

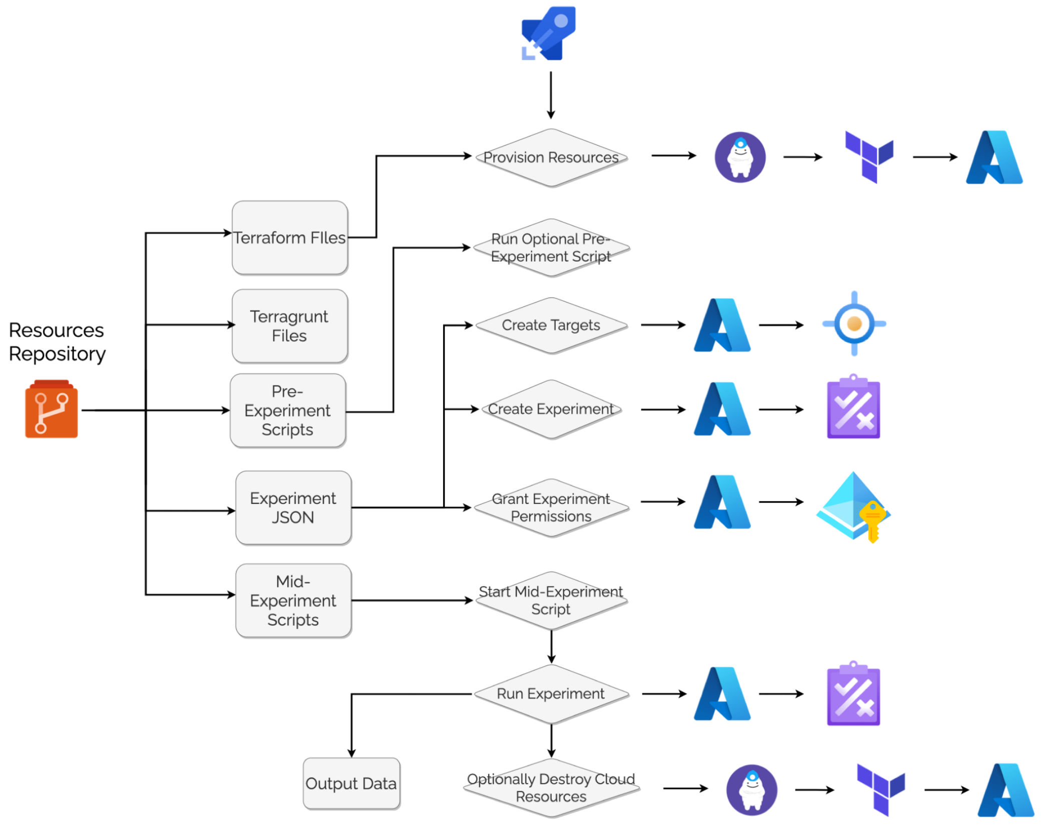


Figure 6.3 Automated Chaos Engineering Process

The above diagram shows the process automated by our framework. After being invoked by an Azure DevOps pipeline and provided with IaC and experiment-defining JSON, the container will provision the resources and conduct the requested experiments while running any data collection scripts in parallel. The container will monitor the experiments, deliver the resulting data to a specified location, and optionally destroy the resources involved when all is finished.

6.3 Example Chaos Framework Scenarios

Having identified the environments on which to implement chaos engineering, we established a list of potential fault scenarios and diagrammed the potential effects of them.

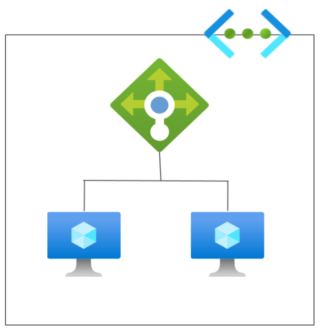


Figure 6.4 Basic Resource Allocation

Figure 6.4 outlines a scalable network structure, where at the simplest level a single load balancer distributes network traffic between two independent virtual machines. Using the icons below in Figure 6.5, we demonstrated what resources would be impacted in each different fault scenario. Modeling these situations will help State Street in the modernization of their critical infrastructures by establishing chaos engineering practices that help identify problems early in the cycle of application deployment. For more details related to the specific causes, effects, and value of each test case refer to Appendix E which contains the final project deliverable.

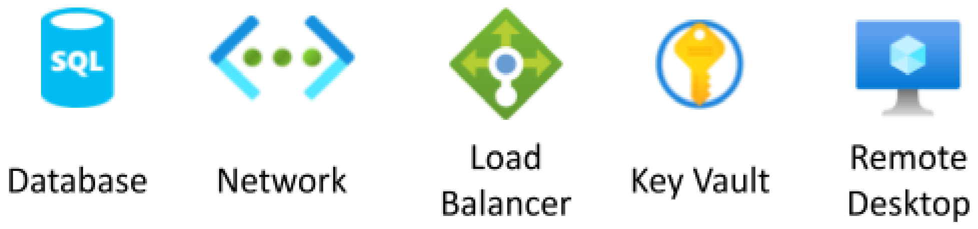


Figure 6.5 Chaos Scenario Resource Icons

Test Case 1 - Outages

Description: A given resource or resource experiences a period of downtime where services are unavailable as a result of unexpected circumstances.

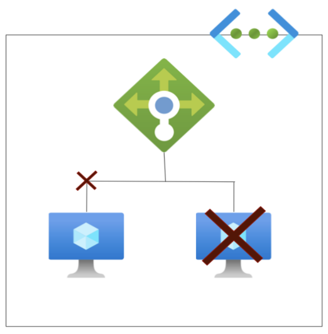


Figure 6.6 Virtual Machine Shutdown or Network Disconnect Scenario

Test Case 2 - Resource Stress

Description: Resources experience a higher load than expected on their internal systems

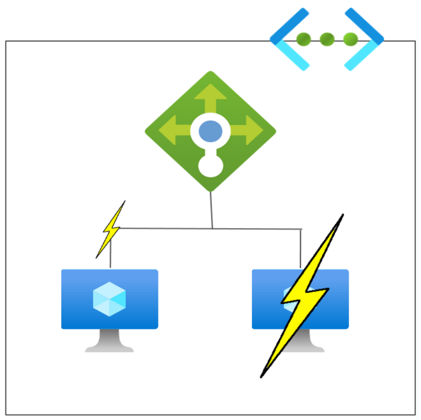


Figure 6.7 CPU Overload or Virtual Memory Pressure Scenario

Test Case 3 - Internal Failures

Description: Processes or operations on a resource fail

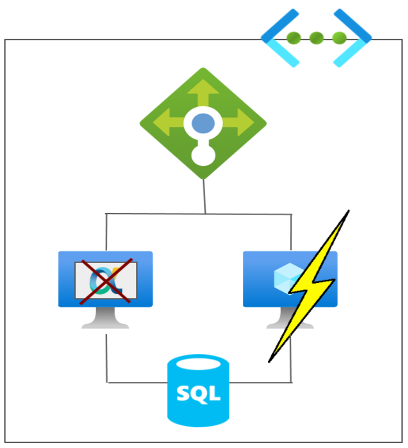


Figure 6.8 Killed Process Scenario

Test Case 4 - Security

Description: A change in access credentials occurs

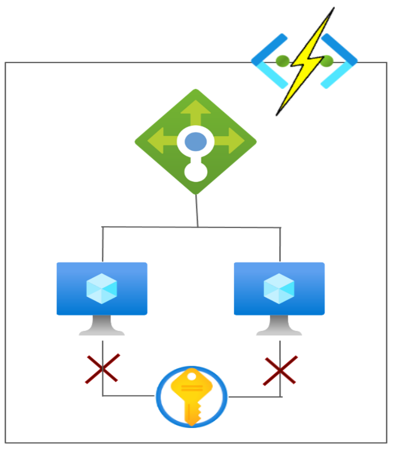


Figure 6.9 Secure Access Management Scenario

7. Software Development

Due to the timeline and nature of our project, we opted to follow a specialized agile scrum with one-week sprints spanning from Monday to Friday. Our schedule consisted of daily scrum around noon, daily sponsor meetings at one o'clock, weekly sprint planning on Monday mornings, and sprint retrospectives on Friday evenings. Within our daily team standups, each member discussed what they were able to accomplish the previous day, what they intended to complete throughout the course of the day, and any blockers that they had encountered. Within our daily sponsor meetings, we met with our primary contact Prafull and discussed the current status of specific objectives, assessed the overall project direction, and addressed any tasks we needed assistance with or additional software permissions for. These daily meetings enabled us to quickly adapt to any changing information and requirements. During our sprint planning meetings, we worked to dictate user stories for the week, estimate their weight, and designate them to members based on best fit. Our retrospective meetings detailed the accomplishments of our team, challenges and potential solutions, and upcoming priorities. In our first week of work, we also attended backlog grooming meetings and daily standup with the larger State Street team to gain an idea of how they implement the agile scrum methodology. In our last week we also presented a formal demo of our code and work to the company for review.

Monday	**Tuesday**	**Wednesday**	**Thursday**	**Friday**
Sprint Planning			Standup	
Standup	Standup	Standup	Advisor Check-In	Standup
Sponsor Check-In	Sponsor Check-In	Sponsor Check-In	Sponsor Check-In	Sponsor Check-In
				Sprint Retrospective

Figure 7.1 Weekly Meeting Overview

Over the course of the term, we completed four main sprints, with an additional acclimation sprint in the beginning relating to onboarding activities and familiarizing ourselves with the existing infrastructure and two buffer sprints at the end for any overflow work. For our sprint tables below we included our user tasks with their respective weekly status, point value, and assignee, as well as the epics that encapsulated these tasks. In order for accurate reporting of work completed, we equated one hour to one story point and as discussed in Section 4.1, we used Jira boards to coordinate and organize these stories.

7.1 Sprint 0: Onboarding and Acclimation

7.1.1 Summary

For our first sprint, our team's main focus was to establish a strong foundation for the technologies we would be working with for the duration of our project. We spent the majority of the first couple days familiarizing ourselves with the remote environment and completing mandatory company onboarding modules to ensure that our access to softwares would not be impacted later in the project. Throughout this week, we determined and maintained a regular meeting cadence with our sponsors and were able to sit in on some of the larger team's discussions to gain a better understanding of their organization and best practices. During this sprint we also worked to more clearly outline a minimum viable product and determine our project's scope so that we could begin creating detailed user stories accordingly.

7.1.2 Documentation

Status	Story Owner	User Story	Points
Epic: Onboarding Activities			
Completed	N. Lopez	State Street Required Leanings	5
Completed	R. Sawka	State Street Required Leanings	5
Epic: Chaos Engineering Background			
Completed	All	Develop project scope	4
Completed	N. Lopez, R. Sawka	Manually run VM outage chaos experiment	4
Epic: Paper Reporting			
Completed	N. Lopez	Update intro goal	0.25
Completed	N. Lopez	Write background paragraphs	4
Completed	R. Sawka	Write Chapter 4.1	3
Completed	O. Scola	Edit Chapter 3	1
Carried Over	O. Scola	Write background risk sections	10
Total Points Completed (Maximum: 36.25)			26.25

Table 7.1 Sprint 0 User Stories

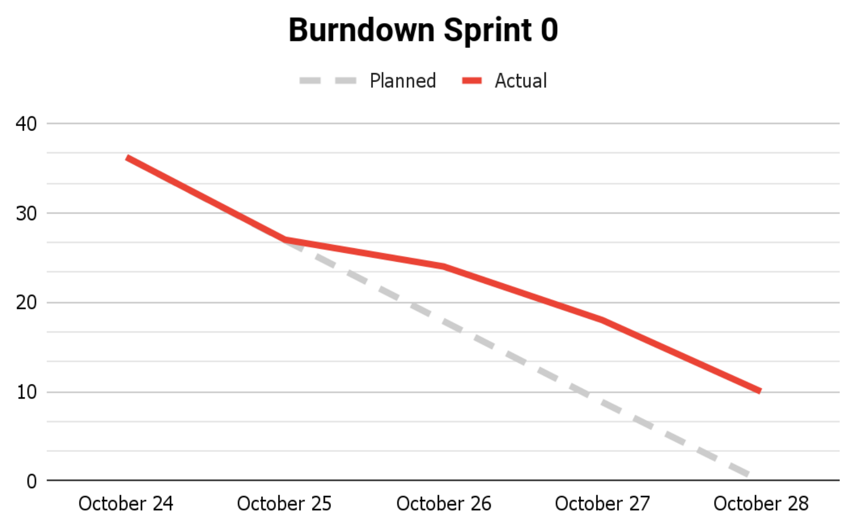


Figure 7.2 Sprint 0 Burndown Chart

7.1.3 Retrospective

While we had initially anticipated being able to produce a unit of work by the end of this week, this sprint was our acclimation period and was primarily targeted towards redefining our final deliverable to be a feasible and valuable objective. Outside of completing generic onboarding, we gained approval of our MVP, scoped out additional work, and began requesting access permissions to required domains and certain features. One of the major challenges that we encountered during this sprint was the pivot in the definition of our project goals, which led to a delay in experimenting with the technologies as new layers of abstraction were added. Outside of this, some improvements that we identified could be made in future sprints were ensuring we keep our advisors regularly updates on our project developments beyond our weekly meetings, planning more extensively with Jira boards and user stories, and placing a stronger emphasis on the paper.

7.2 Sprint 1: Exploration of ADO Pipeline

7.2.1 Summary

At the conclusion of the previous sprint we were successful in manually executing an experiment in Azure Chaos Studio and during this sprint, our team focused on creating a pipeline in Azure DevOps that would prompt the provisioning of resources in the cloud to target in these experiments. We spent the majority of the week researching, experimenting, and developing this pipeline that will handle much of the automation that we had designated for the final stages of our deliverable.

7.2.2 Documentation

Status	Story Owner	User Story	Points
Epic: Pipeline Templating			
Completed	N. Lopez, R. Sawka	Debug permission errors in creating cloud resources in Azure portal	8
Completed	N. Lopez, R. Sawka	Follow IaC developer workflow to avoid authorization blocks	10
Completed	N. Lopez, R. Sawka	Identify virtual machine image IaC	2
Completed	N. Lopez, R. Sawka	Procure correct VM image developer workflow to avoid authorization blocks	7
Completed	N. Lopez, R. Sawka	Procure working virtual network and subnet	3
Completed	N. Lopez	Configure remote Terragrunt state	3
Completed	N. Lopez	Naming modifications to plateng pipeline	4
Completed	N. Lopez	Debug missing PubSub provider	2
Completed	N. Lopez	Add AzureRM provider with PubSub alias to Terragrunt configuration	1
Epic: Paper Reporting			
Carried Over	N. Lopez	Augment Chapter 4.3	1
Completed	N. Lopez, R. Sawka	Construct chapter 6 architecture diagrams	1
Completed	R. Sawka	Write Chapter 7 introduction	2
Completed	R. Sawka	Write Chapter 7.1, 7.2	3
Completed	R. Sawka	Create cover page	0.5
Completed	O. Scola	Edit chapter 2	1
Completed	O. Scola	Edit chapter 4	0.5
Carried Over	O. Scola	Write background risk sections	10
Total Points Completed (Maximum: 59)			48

Table 7.2 Sprint 1 User Stories

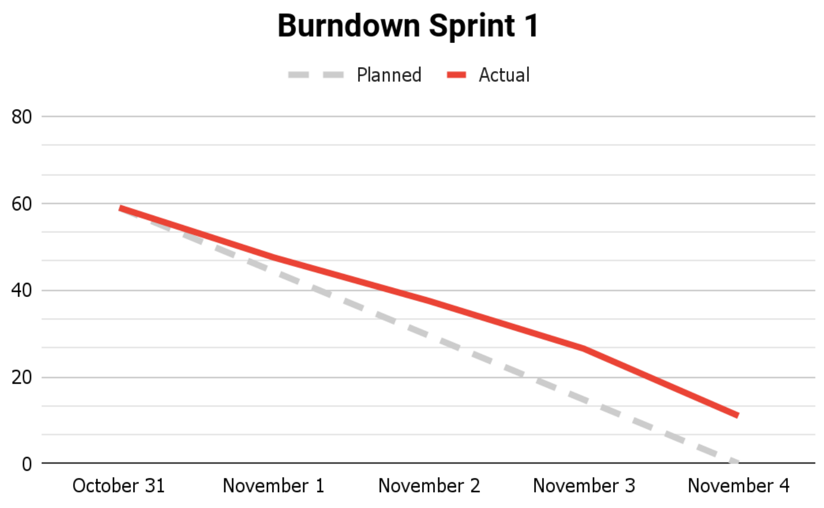


Figure 7.3 Sprint 1 Burndown Chart

7.2.3 Retrospective

During our first working sprint, we were able to holistically refine what we intended on accomplishing by the end of the term and determined that to keep in line with company best practices we would utilize pipelines for resource provisioning automation. A large fraction of the work we completed this week was dedicated to experimenting with the Azure DevOps pipeline and Terragrunt and conducting research related to this tool. We established better communication with our sponsors and other contacts at State Street who helped us create our pipeline specialized for the Pre Dev environment. We also scoped out additional work opportunities past the official MVP in the event that we are able to finish our experiments early. This primary obstacle that we were faced with this week was the significant amount of new learning that was required in order to effectively use the additional tools of ADO and Terragrunt, which also led to some associated problems and access issues. This week we also had minor struggles related to team availability as external time commitments hindered our ability to learn more about the business incentives and considerations. The areas that we cited for improvement in future sprints were in maintaining more structured project management and board organization and analyzing the financial and business aspects.

7.3 Sprint 2: IaC Developer Workflow Configuration

7.3.1 Summary

This week our teams' efforts were a continuation of the previous week's initiative, targeted towards applying the IaC developer workflow we were provided in the company Confluence pages. Similarly to the last sprint, this process involved iterative error discovery and resolution as we looked to convert the platform-engineering pipeline to be functional within the Pre-Dev environment.

7.3.2 Documentation

Status	Story Owner	User Story	Points
Epic: Pipeline Templating			
Completed	N. Lopez	Configure correct DNS zone for key vault's private endpoints	2
Completed	N. Lopez	Identify primary and secondary spoke virtual networks	1
Completed	N. Lopez	Create primary and secondary subnets	1
Completed	N. Lopez	Identify primary and secondary hub virtual networks	1
Completed	N. Lopez	Identify DNS zones inside hub networks	1
Completed	N. Lopez, R. Sawka	Resolve key vault creation conflict in application	3
Completed	N. Lopez, R. Sawka	Identify Terraform pipeline's missing key vault service principle from the logs	1
Discontinued	N. Lopez, R. Sawka	Debug key rotation policy error on applying new DES key	2
Epic: Experiment Data Collection Tools			
Completed	N. Lopez, R. Sawka	HTTP Request Server Development	0.5
Completed	N. Lopez, R. Sawka	HTTP Request Client Development	0.5
Completed	N. Lopez, R. Sawka	Server and Client Integration, Data Collection	3
Epic: Paper Reporting			
Completed	R. Sawka	Write Chapter 4 introduction, Chapter 4.2, 4.3	4
Completed	R. Sawka	Write Chapter 5.1, 7.3	2
Completed	O. Scola	Presentation for Monthly Global Team Connect	2
Carried Over	O. Scola	Write background risk sections	10
Total Points Completed (Maximum: 32)			22

Table 7.3 Sprint 2 User Stories

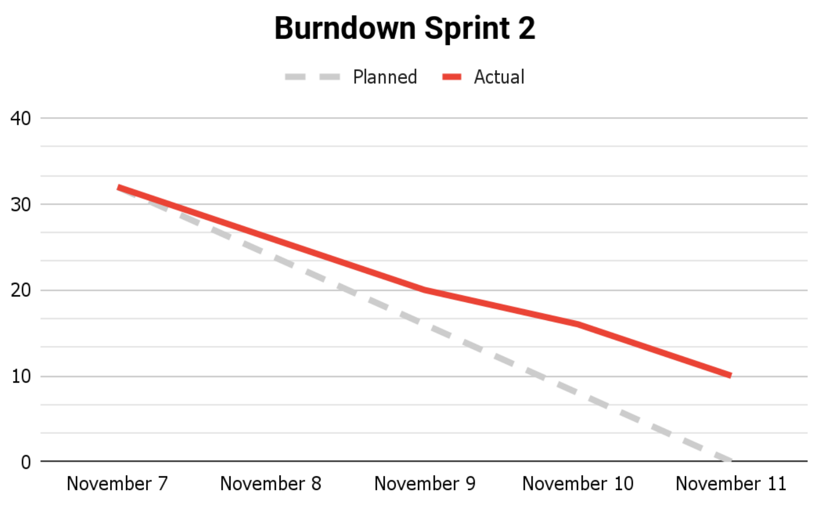


Figure 7.4 Sprint 2 Burndown Chart

7.3.3 Retrospective

Out of the sprints that we had completed up until this point, this second sprint was the one where we encountered the most roadblocks. This week focused on automating the provisioning of the initial environment for our MVP, but unforeseen issues in converting the template IaC to our designated Azure environment caused this process to lag significantly. We were successful in procuring and creating both primary and secondary virtual networks and subnets. We also were able to write the IaC to provision a key vault with both primary and secondary private endpoints. We discovered, however, that modeling the pipeline in State Street's platform engineering Azure subscription created an inadequate environment for our PreDev pipeline. Alongside this, we struggled to find engineers readily available to help us resolve some of the more complicated problems we ran into. A final struggle of ours was the chain of command that we had to traverse as we looked to modify the pipelines since each iteration required new approvals.

As a result of this, we decided to pursue a parallel approach - one that focused on formulating additional scenarios and automating the chaos experiments to be in line with State Street's DevOps provisioning workflow simultaneously. After this change in direction was approved, we worked with our sponsors to adjust permissions so that a manually provisioned environment was created for us to carry out our MVP's initial scenario. Throughout the remainder of the sprint, we prioritized the initial development and testing of a connectivity tool that we will use for data collection. With this renewed focus on chaos engineering and completing our MVP framework, we created user stories related to the manual execution and data collection of the experiments, but due to time constraints, we carried these over to the next sprint. Reflecting on this sprint and looking ahead to the next, we could improve by maintaining more structured agile meetings, being more proactive with our requests for help, and designating our time more effectively toward the tasks that need to be accomplished.

7.4 Sprint 3: Initial Experiment Automation

7.4.1 Summary

With this sprint marking a little beyond halfway through our official sprint weeks, we began to make significant headway on our project deliverables after the blockers that inhibited us from doing so in the previous week. We worked to build the initial automation framework for the chaos experiments and also officially began the experiment phase.

7.4.2 Documentation

Status	Story Owner	User Story	Points
Epic: Chaos Experiments			
Completed	N. Lopez, R. Sawka	Resolved Chaos Studio subscription registration	0.5
Completed	N. Lopez, R. Sawka	Wrote initial deliverable framework	1
Completed	N. Lopez, R. Sawka	Conducted and documented Group 1 experiments	3
Completed	N. Lopez	Automation scripting	6
Completed	N. Lopez, R. Sawka	Debug key rotation policy error	2
Completed	R. Sawka	Conducted and documented Group 2 experiments	2
Carried Over	N. Lopez, R. Sawka	Conducted and documented Group 3 experiments	10
Carried Over	N. Lopez	Conducted and documented Group 4 experiments	5
Epic: Paper Reporting			
Completed	N. Lopez	Write Figure 6.2 description, Chapter 5.2	1
Completed	R. Sawka	Acknowledgments, Chapter 7.4	1.5
Carried Over	O. Scola	Write background risk sections	10
Total Points Completed (Maximum: 42)			17

Table 7.4 Sprint 3 User Stories

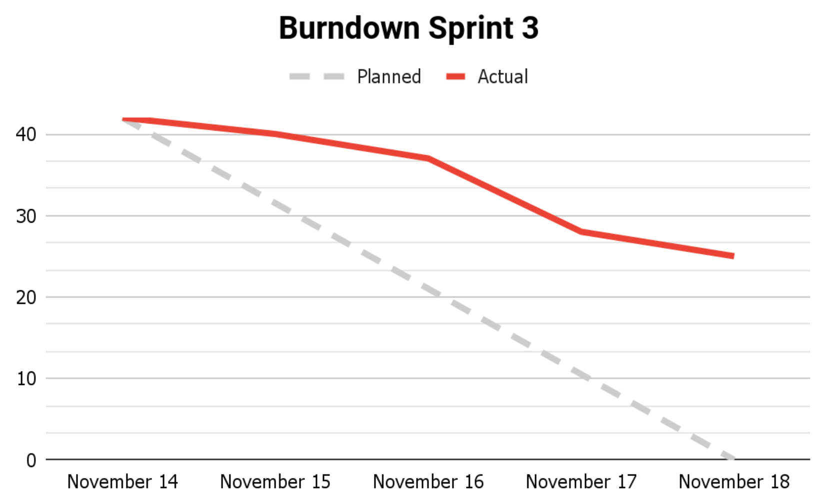


Figure 7.5 Sprint 3 Burndown Chart

7.4.3 Retrospective

Reflecting on this past sprint, our team was able to reach a lot of large milestones and set the groundwork for accomplishing more in the coming sprint. In addition to writing the initial deliverable framework, we were able to establish a docker container that automates the entire chaos engineering process. The container receives IaC and provisions an entire web server environment with multiple web servers on a load balancer. Data is collected while virtual machines in the system are shut down sequentially, measuring changes in request latency. The container then deallocates the whole environment, leaving just the experimental data behind. Alongside this automation we conducted and documented experiment scenarios in groups one and two and made progress on group four. Due to the learning involved with Azure Kubernetes Service for scenarios in group three, we decided to assign that to a later sprint week. We encountered relatively few blockers to our work this week and were actually able to resolve the error from the previous sprint, allowing us to complete a previously discontinued user story. This was largely thanks to support we received from the offshore team members at State Street, although it was sometimes difficult to interface with them due to the significant time difference. For improvements in future sprints, we have acknowledged that proper task reporting within State Street's Jira boards is not something we have been prioritizing and will ensure that it is more thoroughly updated.

7.5 Sprint 4: Kubernetes Fault Automation

7.5.1 Summary

This week effectively marked the conclusion of our official sprints as the conclusion of our project would merely follow us completing our report, preparing a live demonstration to present to our sponsor team, and finalizing our deliverable document. In this final sprint, we wrapped up previous experiment groups and tackled the more complicated Kubernetes experiments that required initial learning. We also prioritized paper development in order to give our contacts at State Street enough time to proofread it for compliance and redactions.

7.5.2 Documentation

Status	Story Owner	User Story	Points
Epic: Chaos Experiments			
Completed	N. Lopez, R. Sawka	Conducted and documented Group 3 experiments	10
Completed	N. Lopez, R. Sawka	MVP deliverable draft	3
Completed	N. Lopez	Conducted and documented Group 4 experiments	5
Completed	N. Lopez	Kubernetes IaC	3
Completed	N. Lopez, R. Sawka	Kubernetes CLI scripting	4
Completed	N. Lopez	Kubernetes pod fault experiment automation	5
Completed	N. Lopez	Generalized script support for service/agent faults	5
Epic: Paper Reporting			
Completed	All	Abstract, Executive Summary, extensive editing	7
Completed	All	Chapter 8, Chapter 9, Chapter 10	5
Completed	N. Lopez	Chapter 4.3.3, 4.4.5	1
Completed	N. Lopez, R. Sawka	Chapter 6 framework diagramming	4
Completed	R. Sawka	Chapter 2.1, 2.2.4, 2.2.5, 2.2.7	2
Completed	R. Sawka	Chapter 7.5	1
Completed	O. Scola	Write background risk sections	10
Total Points Completed (Maximum: 65)			65

Table 7.5 Sprint 4 User Stories

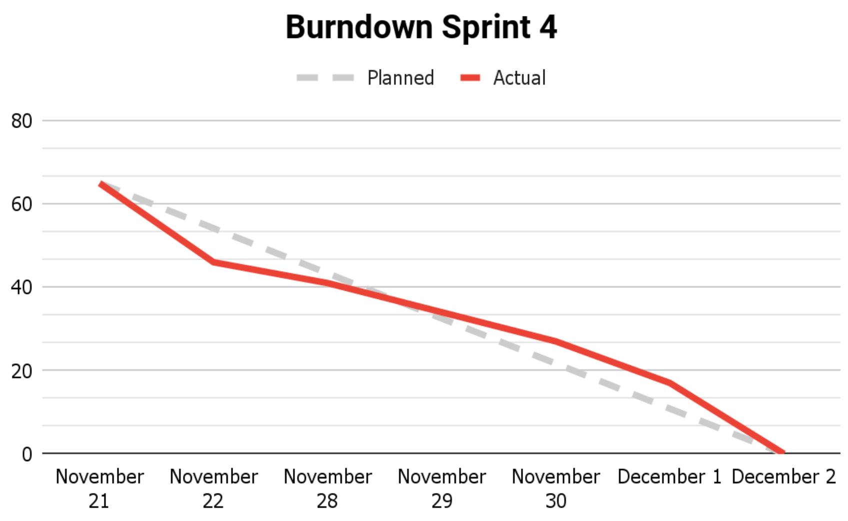


Figure 7.6 Sprint 4 Burndown Chart

7.5.3 Retrospective

In combining the last week and a half of our project work together into one sprint, we undertook a significant amount of story points in order to complete the tasks we set out to accomplish in the beginning of the term. We completed automating the basic fault with Kubernetes and completed a draft of our final deliverable. In preparation for our report publication we also heavily emphasized paper development, working to complete the sections remaining from prior weeks. The challenges we faced this sprint were less technical in nature; some final learnings were needed in order for us to conduct the AKS experiments and with the project deadline closing in we had to strike a balance between finishing our paper and our State Street deliverables. Relating to potential areas of improvement, we could have maintained closer contact with our sponsors, holding code reviews to get more detailed feedback.

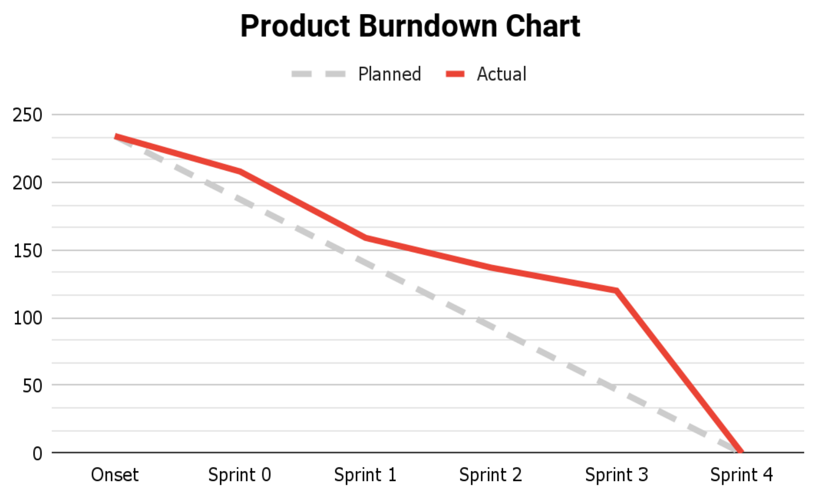


Figure 7.7 Product Burndown Chart

8. Assessment

8.1 Business Learnings

Working with State Street enabled the group to fully immerse in the world of corporate finance, and gave us many insights into industry culture, operations, and team project management. We were able to utilize the knowledge we gained throughout the term to develop a meaningful project and build connections within the company.

Although our project resulted in significant accomplishments, creating a thorough project plan earlier in the timeline would have enabled us to do even more. We struggled in both the pre-MQP term and the beginning of the project to find a clear direction for our work. Collaborating with a large financial company came with communication problems and we had a difficult time deciding where to begin the project planning. After building a stronger relationship with our sponsors, we began asking more questions and really developing our MVP.

Once we officially began the project term, one of the first skills we had to familiarize ourselves with was acknowledging when we collectively needed to ask for help. In retrospect, we realized that our team should have established a meeting cadence in anticipation of technical difficulties instead of causing delays by waiting for them to arise before reaching out to our sponsors. Once we began to maintain a communication pattern throughout daily standup meetings, our sponsors were willing and able to pull in resources to help us overcome our obstacles. A large component of the State Street culture centers around collaboration, encouraging employees to lend their technical expertise to coworkers and teams in need, and this eagerness to help, is something that aided us heavily in the completion of our project.

We were given the opportunity to work with some of State Street's international team members, including our team's contacts in India, who played integral roles in our project and were major contributors to our success. Resulting challenges did arise, however, as those best suited to help us technically were in a completely different time zone, leading us to make accommodations on our end in order to not further limit overlapping availability. State Street has a robust set of checks and balances and although, in the long run, this is the most effective way to securely integrate new technological developments, we as a team found that some of these regulations and red tape slowed down our workflow. It was only when we took a step back and looked at the root of the problem that we began to discover solutions.

As far as work conditions, our team found it very manageable to work remotely alongside State Street. We found it easy to communicate over zoom and messaging platforms, in addition to booking online campus space for our immediate project team as needed. The largest challenge we faced with the remote work culture was interfacing with our sponsors without direct onboarding and environment access. Remote technical work was only sometimes difficult because we could not just walk over to someone's cubicle and get assistance, but rather needed to formally schedule a meeting.

State Street’s communication environment was strictly isolated to their network, which required us to log on to VDI’s to directly message our sponsors. This made it less convenient to ask for help immediately, but after discovering that the majority of our team was working remotely, we quickly adapted to better leverage our remote environment to reach out directly to the team members with the necessary technical knowledge. Working within State Street’s VDI’s slowed down both our project work and communication, which caused us to reevaluate our work environments to avoid environment lagging.

While trying to fix smaller problems in the project, there were times where we failed to take a step back and acknowledge the larger picture. We spent a lot of time dealing with strict IaC modules to provision environments within the corporate network, when we should have had at least one team member designing basic proof-of-concepts outside the network where there are no time-consuming restrictions. Working on a team of three required flexibility of roles of work, but the team gained a significant amount of knowledge in team management and the division of work. One of the most significant improvements we could have made was following a more rigid meeting schedule with allotted in person work time for sprint meetings and team collaboration.

The team made large strides in this project, accomplishing not only a viable end deliverable for our sponsors but also significant levels of technical learnings on cloud resiliency. In retrospect, there were many learnings that may have enabled us to develop an even valuable project, but were left untouched due to time constraints. Looking back, there were many technical concepts that needed to be learned as foundational knowledge in order to be successful. If more of the specific environments and technical knowledge involving Azure was shared within the team at an earlier date, the timeline of work could have been shifted, allotting more time for tangible development.

We were able to be so efficient in relation to the learning curve of these new technologies due in part to the company culture of work. As our team encountered roadblocks, our sponsors were able to call in team members and outside sources to assist us. Our overall learning from this corporate experience was that as the field expands into new technologies, requiring high levels of new learning, a cooperative work environment enhances productivity.

8.2 Technical Learnings

Azure Resources

In order to provision and experiment on the environments we wanted, we needed to understand the resources and services offered by Azure. We had to learn how to create these resources and how to configure them such that they could interact with each other in a secure and meaningful way. Some of these Azure resources are listed below:

- Virtual Machines
- Virtual Networks
- Network Security Groups
- Private Endpoints
- Key Vaults
- Virtual Machine Images
- Azure Kubernetes Service
- Storage Accounts
- Private DNS Zones

Learning these resources will enable us to wield these Azure tools and adapt them to future problems. We now understand how many cloud environments are structured and will easily be able to understand how future environments behave.

Azure Devops Pipelines

State Streets IaC is provisioned through Azure DevOps Pipelines, so we needed to learn how to configure pipelines of our own to provision our environments. We also needed to understand how our framework could be invoked from these pipelines so that chaos experiments could be entirely automated. This required the following learnings:

- Creating Agents
- Configuring Agent pools
- Writing pipeline YAML
- Creating compatible container images
- Configuring agents to pull IaC from private repos

Learning DevOps pipelines will empower us to automate testing and deployment in future development environments. We now understand many fundamental DevOps concepts and use cases and will be able to recognize when they can be applied.

Azure Active Directory

Almost everything we worked with involved some understanding of Azure's enterprise identity service, Active Directory. Especially in a multi-subscription hierarchy like the State Street network, we needed an understanding of the following Active Directory concepts in order to ensure our resources could interact with each other with minimum priviledges.

- Subscriptions
- Service Principles
- User Managed Identities
- RBAC Configuration

We can use these concepts to accelerate our learning of Active Directory and other identity services, and apply the knowledge to ensure future cloud environments are secure.

Terraform

We provisioned cloud resources with IaC using Terraform. We needed to learn how to use the tool, which involved the following understanding:

- General syntax
- Azure provider syntax
- Module creation/use
- Remote state storage configuration
- Forking and fixing Azure open source terraform modules

With knowledge of IaC tools like Terraform, we can provision any cloud environment in minutes. We can use these skills to automate development, testing, and deployment of cloud applications in the future.

Network Security

When using State Streets IaC modules to provision our environments, we had to adhere to the strict network security practices they enforced. This involved an understanding of the following concepts, so that we could provide the modules with proper inputs.

- State Street’s subscription hierarchy
- State Street’s virtual networks and subnets
- Primary/Secondary Endpoints
- Primary/Secondary DNS zones
- Private Image Galleries
- Hub-Spoke topology
- Key vault RBAC

Now that we understand these network security concepts, we can identify vulnerabilities and bring any future network environments that we work with up to date with modern security standards, keeping them safe from mistakes and malicious actors.

Azure CLI

Our framework automates many interactions with the Azure Resource Manager through Azure CLI. To ensure this interaction was reliable, we needed to understand the variations of the following Azure CLI operations:

- Authentication
- Creating resources
- Querying information
- Granting permissions
- REST API resource manipulation

We can use Azure CLI to control any of our future Azure environments. Independent of changing graphical interfaces, we can execute commands in headless environments, integrate them into scripts, and professionally document the entire process with text.

Chaos Engineering Concepts

We needed to understand the following chaos engineering concepts in order to plan out our framework, and make sure that the most important capabilities were covered early on.

- Outages
- Overloads
- Access Denials

We can now conduct chaos engineering on any future cloud environment we work with to ensure that they are prepared for disaster.

Azure Chaos Studio

Our framework is centered around a Chaos Studio, a new Azure feature with little documentation and limited CLI support. We had to figure out if it was even possible to automate these tools, along with the following concepts:

- Resource Targets
- Service-based Faults
- Agent-based Faults
- Experiment JSON Syntax
- Experiment RBAC
- Azure REST API Calls

Learning Azure Chaos Studio made it easy for us to conduct and automate chaos engineering in any future Azure environment we work with. This has also given us a list of things to test when chaos engineering, whether or not we are working with Azure resources.

Kubernetes

Some of the system failures supported by our framework involved Azure Kubernetes Service, requiring our understanding of Kubernetes in order to design and test them.

- Nodes
- Pods
- Services
- Use Cases
- KubeCTL API
- Manifest YAML

With Kubernetes, we can combine disparate computing resources into a single machine for containerized applications. This has many uses in distributed computing applications, including modern microservices. We can now wield Kubernetes, allowing us to work with any future applications that use it.

Docker

Our framework was designed inside a docker container for several reasons. In order for this to be possible, we needed a general understanding of docker.

- Docker API
- Dockerfile Syntax

With docker, we can develop applications for any environment, from any computer. This knowledge is necessary in our future careers because very rarely will we be developing applications purely for our own operating system.

Common Cloud Use Cases

We wanted our framework to be accompanied by several relevant examples. We needed to understand the following common use cases for the cloud resources Azure provides.

- Load Balanced VM Web Servers
- Kubernetes Web Server / Database environment

With this knowledge, we understand many of the uses and justification for the cloud and its tools. When faced with a computing task, we can determine whether or not cloud computing is applicable, and apply it if it is.

VM Remote Control

Much of our framework was tested inside a VM for RBAC purposes, and many of our examples involved environments with VMs. To make these VMs behave, we needed an understanding of remote control over SSH, as well as basic Linux knowledge.

- SSH configurations
- Linux user/permissions management

Virtual machines can be utilized for innumerable applications, but only if you know how to control them. In our future careers, we can be confident that we can interact with remote machines securely.

Bash Scripting

Since many of the tools we used work through the CLI, bash scripting handled much of the automation. This required knowledge of how to write safe and reliable bash. Almost all modern computers can be controlled with bash. In our future careers, we can use this learning to write scripts for countless use cases.

Python Scripting

The heart of our framework is a python script that sets up an Azure Chaos Studio experiment from its defining JSON. In order to write this, we needed to understand several concepts, including using python to parse JSON and make system calls. Python is one of the most useful programming languages in the world today, with a massive community developing thousands of Python libraries. Knowledge of this tool is essential if we want to take advantage of its countless applications.

8.3 Accomplishments

The team was able to accomplish our goal of creating an automation framework that integrates with Azure Chaos Studio and is easy to use with Azure DevOps pipelines. Along with the completion of the framework itself, the team was able to to document four use cases:

1. Virtual Machine Outages
2. Virtual Machine CPU Pressure
3. Key Vault Access Denial
4. Kubernetes Pod Failure

Each of these use cases were handled by the framework, proving its ability to automate chaos experiments. The steps to replicate these use cases are provided as detailed examples in the framework's documentation so that State Street can quickly learn to adapt the framework to their needs.

8.4 Limitations

The largest and most apparent issue that we encountered repeatedly throughout the project was the time constraint. Restricting a substantial initiative to just around seven weeks greatly limited the scale of our project due to feasibility concerns and even with the narrowed scope, the timeline was rather accelerated. Alongside the already expedited nature of our work, we encountered difficulties and delays with onboarding. As with all large corporations, there are standard onboarding procedures that must be adhered to, however, waiting for these compliance actions to be completed and access permissions to be granted while trying to begin work led to initial impediments. To further exacerbate the problems faced in formally starting our project, the scope of our work was not entirely designed prior to our start date and underwent revisions within the first two weeks. This meant that it was not until our second official sprint that we had a formal requirements document.

Many times throughout the project we also encountered significant delays as a result of needing technical learning. In working with the DevOps pipeline, several layers of abstraction were added which required more learning on our end in order to produce a usable product. Pipeline development was sluggish in nature, requiring approvals from authorized employees for each test, and opportunities for tech support were limited by employees' busy schedules.

While the main goals of the project were accomplished in time, these limitations have left many stones unturned for our framework's possible capabilities.

9. Future Work

Even with a functional automated framework for common chaos scenarios, there remains much to do relating to scalability and inclusivity. Our project laid the groundwork for future adaptations in cloud development and resiliency and the State Street team can further our initiative by more broadly expanding which faults from the Azure action library are executed, as well as which resource types are targeted. A significant remark related to this discussion is that Azure Chaos Studio is mainly accessible to developers in preview mode, requiring State Street to wait for this feature/capability to be stabilized before taking our work and formally establishing chaos engineering best practices for the whole firm. The user manual we created, however, will still guide future developers in their implementations, with the required initial learning and research already completed.

Another potential area for future work involves improving upon the code that we delivered. Many Azure Chaos Studio capability IDs must be added to the scripts and updated as Azure Chaos Studio changes. Perhaps to avoid these updates, the framework could be modified to pull the latest capability IDs directly from Microsoft and use those when parsing experiment JSONs.

Furthermore, the framework is a script and has many inputs, and the verification of these inputs could certainly be improved. The framework could also be modified to parse experiments from inputs other than the strict Azure Resource Manager Template JSON format; perhaps a greatly simplified format could be used so that developers can have more flexibility in how they design experiments with text.

As it currently stands, our framework can run custom data collection scripts during an experiment. These collection scripts run in the same environment as the pipeline agent as a subprocess of the framework. There are certainly better ways to expose the output of custom data collection scripts during experimentation, and that should be explored in order to make the framework more practical. Perhaps running the scripts in a parallel job, or at least showing the output of these scripts in real time through the pipeline's standard output would resolve these concerns.

Finally, developing a method to aid in more efficient experiment JSON file creation would help in any future expansion efforts. The overall direction of the project following our leave will largely be determined by the timeline of other efforts related to this broader company initiative and will be dictated by State Street’s project leads and technology management teams.

10. Conclusion

In partnership with State Street, the project team was able to create a scalable automation framework that utilized Azure DevOps pipelines to conduct chaos experiments, producing data that will expose vulnerabilities in enterprise systems with the goal of reinforcing cloud application resiliency. These experiments address situations of resource unavailability, resource stress, internal failures, and security modifications. Resource provisioning, experimentation and data collection can now be packaged into a single pipeline that can run overnight, outputting the collected data for analysis in the morning.

The developed framework enables State Street to store experiments in text along with the IaC that defines the target resources. When invoked, the framework reads the JSON and prepares experiments, which are then automatically run in parallel with any custom data collection scripts packaged with the JSON. The framework is relatively simple to implement, as experiment-defining JSON files can be automatically generated through the Azure portal, and only a few environment variables need to be declared to invoke the framework via a DevOps pipeline.

While we accomplished the main goals of the project, there remain numerous ways and areas in which our framework can be improved and scaled. As more capabilities are added to Azure Chaos Studio, this framework will become even more useful, opening up new areas of cloud security exploration. In the future, improvements can be made to the framework's overall ease-of-use, such as replacing the experiment JSON format with something more easily parsed, a change that would emphasize the developer experience. With a fully operational proof-of-concept and many opportunities for growth, our framework has a bright future in helping State Street.

With State Street's overarching corporate aim of providing comprehensive and secure financial services, this project enables State Street to gain mobility in reinforcing their internal architecture. Our contributions to the company's chaos engineering practices will provide application reliability for end users, vulnerability identification for State Street engineers, and consumer trust for State Street executives.

References

"Abstract Digital Background with Technology Circuit Board Texture. Electronic Motherboard Illustration. Communication and Engineering Concept. Vector Illustration for Free." Vecteezy. vecteezy, March 28, 2022. https://www.vecteezy.com/vector-art/6826900-abstract-digital-Background-with-technology-circuit-board-texture-electronic-motherboard-illustration-communication-and-engineering-concept-vector-illustration.

Aday S., Andžāns M., Bērziņa-Čerenkova U., Granelli F., Gravelines J., Hills M., Holmstrom M., Klus A., Martinez-Sanchez I., Mattiisen M., Molder H., Morakabati Y., Pamment J., Sari A., Sazonov V., Simons G., Terra J. "2007 cyber attacks on Estonia" Hybrid Threats. A Strategic Communications Perspective. NATO Strategic Communications Centre of Excellence. (2019) https://stratcomcoe.org/publications/hybrid-threats-2007-cyber-attacks-on-estonia/86

Adobe Communications Team. "Waterfall Methodology - a Complete Guide | Adobe Workfront." Adobe Experience Cloud Blog, March 18, 2022. https://business.adobe.com/blog/basics/waterfall.

Alvarenga, Guilherme. "What Is the Shared Responsibility Model?: CrowdStrike." crowdstrike.com, November 14, 2022. https://www.crowdstrike.com/cybersecurity-101/cloud-security/shared-responsibility-model/.

Arzoomanian, R. "A Complete History Of Mainframe Computing." Tom's Hardware. June 26, 2009. https://www.tomshardware.com/picturestory/508-mainframe-computer-history.html

Atlassian. "Agile Project Management Tools for Software Teams." Atlassian 2022a. https://www.atlassian.com/software/jira/agile.

Atlassian. "Confluence: Your Remote-Friendly Team Workspace." Atlassian. 2022b. https://www.atlassian.com/software/confluence.

Barrero. J, Bloom. N, Davis. S. "Why Working from Home Will Stick" National Bureau of Economic Research. April 2021. https://www.nber.org/papers/w28731

Barroso, Luiz André. Clidaras, Jimmy. and Hölzle, Urs. "The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines, Second edition." Synthesis Lectures on Computer Architecture 2013 8:3, 1-154. https://www.morganclaypool.com/doi/epdf/10.2200/S00516ED2V01Y201306CAC024

Beck, Kent, Mike Beedle, Aerie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, et al.. "Principles behind the Agile Manifesto." Agile Manifesto. February 13, 2001. https://agilemanifesto.org/principles.html.

Carey, Scott. “What Is Chaos Monkey? Chaos Engineering Explained.” InfoWorld. InfoWorld, May 13, 2020. https://www.infoworld.com/article/3543233/what-is-chaos-monkey-chaos-engineering-explained.html

Casey, Kela. “Everything You Need to Know about Scrum Methodology.” CODERSERA, January 6, 2022. https://codersera.com/blog/what-is-scrum-methodology/.

Charboneau, Tyler. “What Is Devops?” Orange Matter, October 12, 2022. https://orangematter.solarwinds.com/2022/03/21/what-is-devops/

Chcom. “State Street Corporation.” CompaniesHistory.com - The largest companies and brands in the world, November 20, 2020. https://www.companieshistory.com/state-street-corporation/

“Cloud Controls Matrix and CAIQ v4.” Cloud Security Alliance. June 7, 2021. https://cloudsecurityalliance.org/artifacts/cloud-controls-matrix-v4/

Codecademy. “What Is an IDE?” Codecademy. Codecademy. 2022. https://www.codecademy.com/article/what-is-an-ide.

Digite. “What Is Scrum Methodology? & Scrum Project Management.” Digite, September 30, 2019. https://www.digite.com/agile/scrum-methodology/.

Ebert. C, Gallardo. G, Hernantes. J, and Serrano. N. "DevOps." IEEE Software, vol. 33, no. 3, pp. 94-100, May-June 2016, doi: 10.1109/MS.2016.68. https://ieeexplore.ieee.org/abstract/document/7458761

Finlay, Emily. “Your Introductory Guide to the Waterfall Methodology.” MindManager Blog, August 31, 2022. https://blog.mindmanager.com/waterfall-methodology/.

FitzMacken, Tom. “Azure Resource Manager Overview - Azure Resource Manager.” Azure Resource Manager overview - Azure Resource Manager | Microsoft Learn. 2022. https://learn.microsoft.com/en-us/azure/azure-resource-manager/management/overview.

Globaldata “State Street - Digital Transformation Strategies.” Market Research Reports & Consulting | GlobalData UK Ltd., May 20, 2022. https://www.globaldata.com/store/report/state-street-enterprise-tech-analysis/.

Gunja, Saif. “What Is Chaos Engineering?” Dynatrace news, November 11, 2022. https://www.dynatrace.com/news/blog/what-is-chaos-engineering/.

Haque, Shahla. “Charles River Development to Be Acquired by State Street Corporation: Charles River Development.” Charles River Development, Investment Management Solutions, 2018. https://www.crd.com/news/press-releases/2018/charles-river-development-to-be-acquired-by-state-street-corporation/.

Horner, Matthew and Hyslip, Thomas "SQL Injection: The Longest Running Sequel in Programming History," Journal of Digital Forensics, Security and Law: Vol. 12 , Article 10. June 30, 2017. https://doi.org/10.15394/jdfsl.2017.1475

House, Travor. “10 Top Cloud Providers in 2022 - Allcode - AWS Partner.” AllCode, January 19, 2021. https://allcode.com/cloud-providers/

IBISWorld. “Internet Traffic Volume.” IBISWorld, August 11, 2022. https://www.ibisworld.com/us/bed/internet-traffic-volume/88089/

Isaiah. “An Introduction to Azure Chaos Studio.” Microsoft Learn. Dec 17, 2021,https://learn.microsoft.com/en-us/shows/azure-friday/an-introduction-to-azure-chaos-studio

Kaur, Amanpreet & Kaur, Bikrampal & Singh, Dheerendra. “Optimization Techniques for Resource Provisioning and Load Balancing in Cloud Environment: A Review.” International Journal of Information Engineering and Electronic Business. 9. 28-35. 10.5815/ijieeb.2017.01.04 https://www.researchgate.net/publication/312476290_Optimization_Techniques_for_Resource_Provisioning_and_Load_Balancing_in_Cloud_Environment_A_Review

Kubernetes (software) “Production-Grade Container Orchestration.”. Cloud Native Computing Foundation, the Linux Foundation. 2022. https://kubernetes.io/

Kumar, Anil, Vasudevan Swaminathan, Venkatesh Veerachamy. “Software Development Life Cycle: What Is SDLC?” Zuci Systems. June 21, 2021. https://www.zucisystems.com/services/softwaredevelopment-life-cycle/.

LaunchRack. “IAC and Terraform.” LaunchRack. Accessed November 30, 2022. https://launchrack.com/blog/iac-and-terraform/

Lyudvig, Anna. Traders Magazine. “Best New Product: State Street Alpha Data Platform.” Markets Media, May 27, 2022. https://www.marketsmedia.com/best-new-product-state-street-alpha-data-platform/

Merron, Dan, and Shanika Wickramasinghe. “Infrastructure as Code (IAC): The Complete Beginner's Guide.” BMC Blogs, November 5, 2021. https://www.bmc.com/blogs/infrastructure-as-code/

M.G. Avram. “Advantages and Challenges of Adopting Cloud Computing from an Enterprise Perspective”, Procedia Technology, Volume 12, 2014, Pages 529-534, ISSN 2212-0173. https://doi.org/10.1016/j.protcy.2013.12.525

Microsoft. “About Azure Key Vault”. Azure Security Documentation. October 10, 2022 https://learn.microsoft.com/en-us/azure/key-vault/general/overview

Microsoft. “Hub-spoke network topology in Azure”. Azure Architecture Documentation”. Dec 7, 2022.https://learn.microsoft.com/en-us/azure/architecture/reference-architectures/hybrid-networking/hub-spoke?tabs=cli

Microsoft. “Visual Studio Code - Code Editing. Redefined.” RSS. Microsoft, November 3, 2021. https://code.visualstudio.com/

Mijacobs. “What Is Continuous Delivery? - Azure DevOps.” Azure DevOps | Microsoft Learn. 2022. https://learn.microsoft.com/en-us/devops/deliver/what-is-continuous-delivery

O'Hanley, Ronald. “A Message to Our Stakeholders.” State Street - 2021, 2022. https://annualreport.statestreet.com/Y2021/Details/2022/achieving-our-strategy/default.aspx.

Parsons, Joe. “State Street Leapfrogs BNY Mellon as World's Largest Custodian.” Global Custodian, September 27, 2018. https://www.globalcustodian.com/state-street-leapfrogs-bny-mellon-worlds-largest-custodian/.

Schwaber, Ken, and Jeff Sutherland. “The 2020 Scrum GUIDE.” Scrum Guide | Scrum Guides, 2020. https://scrumguides.org/scrum-guide.html.

Segura, Elvis. “Declarative vs. Imperative in IAC.” Linode, June 1, 2021.https://www.linode.com/blog/devops/declarative-vs-imperative-in-iac/

Shown, Shane. “How to Become an Devops Engineer in 2020.” Medium. The Startup, September 29, 2020. https://medium.com/swlh/how-to-become-an-devops-engineer-in-2020-80b8740d5a52.

Strack, Ben. “State Street Sees 'Significant Opportunity' in Tokenization.” Blockworks, August 29, 2022. https://blockworks.co/news/state-street-sees-significant-opportunity-in-tokenization/

State Street Azure Platform Engineering, “State Street Project Scope”. State Street, 2022.

Tafoya, Fernanda. “What Is Cloud Adoption and Why Is It Important?” iTexico. June 3, 2020. https://www.itexico.com/blog/what-is-cloud-adoption

“The 5 Main Advantages of Cloud Computing in the Smart Working Regime.” LogicalDOC, January 22, 2021. https://www.logicaldoc.com/blog/521-the-5-main-advantages-of-cloud-computing-in-the-smart-working-regime

Trevino, Becky. “How Digital Transformation Impacts Technical Skills Gap.” Digital Transformation Impact on the Technical Skills Gap | Spiceworks 1, October 12, 2022. https://www.spiceworks.com/tech/it-careers-skills/guest-article/how-digital-transformation-impact-technical-skills-gap/

Wright, Lucy. “Azure vs AWS: Why People Are Choosing the Microsoft Cloud.” Core Technology Systems, 2019. https://www.core.co.uk/blog/azure-vs-aws-why-people-are-choosing-the-microsoft-cloud

Appendix A: Glossary of Terms

Agile Scrum

A project management system that emphasizes incremental and iterative development.

Azure Key Vault

A cloud service for securely storing and accessing cryptographic keys i.e. secrets.

Chaos Engineering

Testing a system's ability to withstand disruptions with minimal consumer impact.

Cloud

Software and services that are hosted and accessed over the internet rather than locally.

Cloud Service Provider (CSP)

A company that provides computing resources, infrastructure and software as a service.

Command Line Interface (CLI)

A text-based interface that receives user commands and performs those operations.

Continuous Delivery (CD)

The practice of using automation to accelerate software production and code releases.

DevOps

The combination of IT operations and software development tools and practices.

Financial Technology (FinTech)

Software and applications created to improve and automate financial services.

Infrastructure as Code (IaC)

The management and provisioning of infrastructure through code instead of manually.

Kubernetes

A container orchestration system for automating application deployment and management.

Software Service Level Agreement (SLA)

A contract between a service provider and the customer that outlines acceptable levels of required services.

Sprint

A short period of time where a team works to complete a designated amount of work.

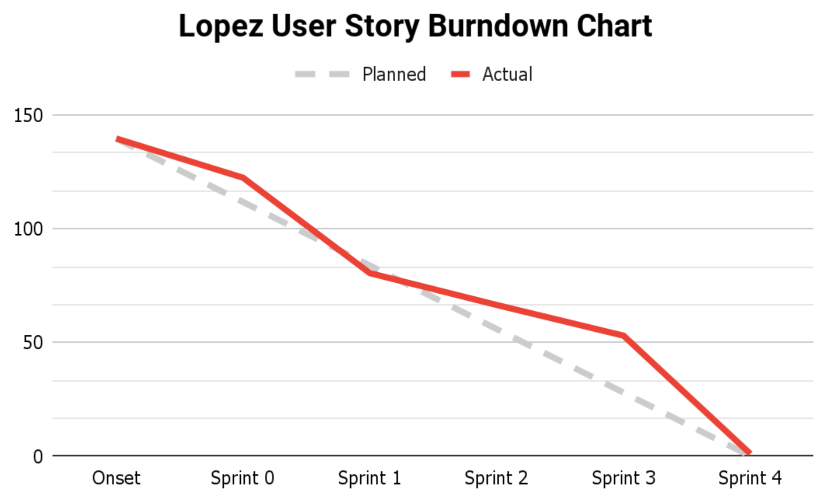
Appendix B: Gantt Chart

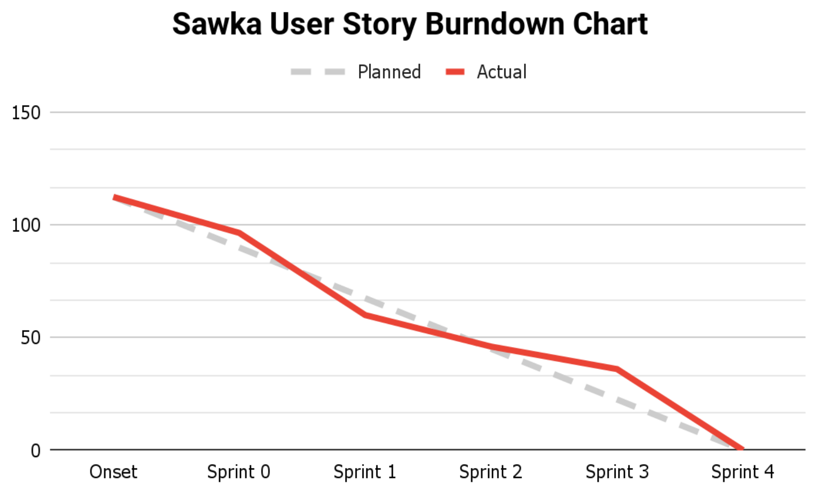
PROJECT TIMELINE

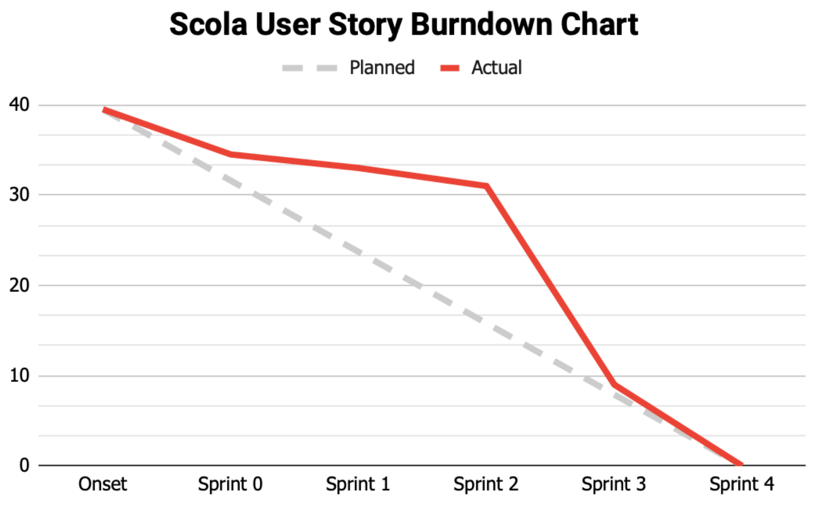
PROJECT TITLE	Azure Chaos Studio	COMPANY NAME	State Street
PROJECT MANAGER	Prafull Srivastava and Ravi Palanki	DATE	10/24-12/16

	Project Weeks:		OCT 24-28	OCT 31-NOV 4	NOV 7-11	NOV 14-18	NOV 21-DEC 2	DEC 5 - DEC 13
	Sprints:		0	1	2	3	4	5 (Burndown)
1	Onboarding Activities	MVP	🟦	🟦	🟦		🟦	▨
		State Street Required Learnings	🟦	🟦				▨
2	Chaos Engineering Background	Develop Project Scope	🟦					▨
		Manually run VM outage chaos experiment	🟦					▨
3	Pipeline Templating	Debug permission errors in creating cloud resources in Azure portal		🟥				▨
		Follow IaC developer workflow to avoid authorization blocks		🟥				▨
		Identify virtual machine image IaC		🟥				▨
		Procure correct VM image developer workflow to avoid authorization blocks		🟥				▨
		Procure working virtual network and subnet		🟥				▨
		Configure remote Terragrunt state		🟥				▨
		Naming modifications to plateng pipeline		🟥				▨
		Debug missing PubSub provider		🟥				▨
		Add AzureRM provider with PubSub alias to Terragrunt configuration		🟥				▨
		Configure correct DNS zone for key vault's private endpoints			🟥			▨
		Identify primary and secondary spoke virtual networks			🟥			▨
		Create primary and secondary subnets			🟥			▨
		Identify primary and secondary hub virtual networks			🟥			▨
		Identify DNS zones inside hub networks			🟥			▨
		Resolve key vault creation conflict in application			🟥			▨
		Identify Terraform pipeline's missing key vault service principle from the logs			🟥			▨
		Debug key rotation policy error on applying new DES key			🟥			▨
		HTTP Request Server Development			🟥			▨
		HTTP Request Client Development			🟥			▨
4	Experiment Data Collection Tools	Server and Client Integration, Data Collection				🟨		▨
		Resolved Chaos Studio subscription registration				🟨		▨
		Wrote initial deliverable framework				🟨		▨
		Automation scripting				🟨		▨
		Debug key rotation policy error				🟨		▨
5	Test Case 1 - Outages	Write IAC			🟪			▨
		Conducted and documented Group 1 experiments			🟪			▨
		Automation scripting			🟪			▨
6	Test Case 2 - Stress	Write IAC				🟪		▨
		Conducted and documented Group 2 experiments				🟪		▨
		Automation scripting				🟪	🟪	▨
7	Test Case 3 - Pod Failures	Write IAC					🟦	▨
		Conducted and documented Group 3 experiments					🟦	▨
		Automation scripting					🟦	▨
		Kubernetes IaC					🟦	▨
		Kubernetes CLI scripting					🟦	▨
		Kubernetes pod fault experiment automation					🟦	▨
		Generalized script support for service/agent faults					🟦	▨
8	Test Case 4 - Security	Write IAC				🟩		▨
		Conducted and documented Group 4 experiments				🟩		▨
		Automation scripting				🟩	🟩	▨
9	Paper Reporting	Chapter 1		🟧				▨
		Chapter 2		🟧				▨
		Chapter 3			🟧			▨
		Chapter 4			🟧			▨
		Chapter 5				🟧		▨
		Chapter 6				🟧		▨
		Chapter 7	🟧	🟧	🟧	🟧	🟧	▨
		Chapter 8					🟧	🟧
		Chapter 9					🟧	🟧
		Chapter 10					🟧	🟧
		References					🟧	🟧
		Appendices					🟧	🟧
		Edits					🟧	🟧

Appendix C: Developer Story Point Burndown







Appendix D: MVP Proposal

Below is our initially envisioned project, proposed to our sponsors at the beginning of the term. As we learned more about the opportunities and use cases, we modified the project to focus more on automation rather than researching possible scenarios.

1. Vulnerable Systems

Identification of environments that need chaos testing

a. List of known vulnerable systems/environments
b. How to identify vulnerable systems/environments
c. Existing state street systems that need chaos testing

2. Chaos

Possible chaos scenarios of vulnerable systems + consequences

a. Explanation of scenarios and the systems involved
b. Indicators for the more subtle scenarios
c. Possible effects of each scenario

3. Testing

Conducting a chaos experiment (Azure Chaos Studio)

a. How to simulate the relevant scenarios for each system
    i. List the Azure faults that would simulate the scenario
    ii. List the possible variations of fault parameters
b. How to collect relevant data

4. Solutions

How to use chaos data to improve the system.

For each Scenario <X>:

1. Identify Known Vulnerable Environments
2. Explain Scenario
3. Identify possible scenario effects
4. Write terraform to provision vulnerable environment
5. Cost estimate for each environment
6. Run experiment
7. Explain involved azure faults
8. Explain how to set up experiment
9. Explain expected data
10. Explain how to interpret expected data
11. Identify Solutions
12. Explain how to implement solutions
13. Explain how to verify solution

Appendix E: MVP Deliverable

Chaos Engineering Framework User Manual

Overview

This framework consists of a set of scripts that prepare and run Azure Chaos Studio experiments. It is designed to be incorporated into an automated pipeline that will provision Azure resources, expose them to system failures in controlled experiments, and output any collected data. The framework automates many of the tasks needed to prepare a chaos experiment by parsing an experiment-defining JSON file and executing the necessary Azure CLI commands needed to configure the targets, create the experiment, and grant the necessary experiment permissions.

The framework is a set of two script files, a Bash script and a Python script. The Python script takes a single experiment-defining JSON file and performs the necessary steps to prepare the experiment in Azure. The Bash script uses the Python script to prepare and run an arbitrary number of experiments. The Python script is an independent tool and can be used manually or by any other script, while the Bash script is designed to be useful when invoked by an automated pipeline.

Experiment-Defining JSON

ARM Templates are JSON files created and interpreted by the Azure Resource Manager with all the information needed to define a configuration of cloud resources. Azure Chaos Studio Experiments, like any other Azure resource, have a JSON file that is generated when they are created in the Azure portal. Inversely, the Azure Resource Manager can interpret a portion of this JSON to create an experiment through Azure CLI, which is why our framework is designed to interpret the same JSON to automatically handle the many other tasks needed to prepare an experiment.

Capabilities

The framework has the potential to automate experiments that utilize any of the faults provided by Azure Chaos Studio, as long as the information for those faults is added to the lookup tables within the Python script. As new faults are added to Azure Chaos Studio, these lookup tables must be updated with the new information. The process of updating these tables is defined in the “Maintenance and Future Improvements” section of this user manual under “Maintaining the JSON Interpretation Data.”

Currently, the following faults are supported by the framework:

- Virtual Machine Shutdown 1.0
- Virtual Machine CPU Pressure 1.0
- Key Vault Deny Access 1.0
- AKS Pod Chaos 2.1

Warnings

1. Running an experiment causes real system failures in the target resources. There is no simulation happening here, the faults are actually induced in the system and will affect any services running on the target resources.

2. The service principle that grants the Azure DevOps Pipeline permission to access the Azure Resource Manager needs sufficient privileges to create role assignments for the target resources. This is not always present upon the service principles creation, and must be added. These additional privileges can be limited to the scope of the resource group where the target resources are to be created, ensuring that experiments cannot affect the wrong resources.

3. At the time of writing this user manual, the location property must be included in the experiment-defining JSON files provided to the framework. If this location property does not contain the actual location of the experiment and target resources, there will be errors when trying to run the experiment.

4. The framework must run within an authenticated Azure CLI session to work. Also, if running an Azure Kubernetes Service experiment, Helm must be installed so that the framework can install Chaos Mesh on the cluster.

5. For some agent-based experiments, the stress-ng application must be installed on Linux machines. This is not currently handled by the framework and must be installed separately. We recommend that stress-ng is simply included in the image that creates the VMs.

6. Optional data collection scripts are started as a subprocess of the framework's bash script, and sent a SIGTERM to terminate at the end of the experiment. If this is in an Azure DevOps pipeline, the script must be designed to terminate within 10 seconds of this signal. If it does not terminate, the pipeline will not transition to the next task.

setup_experiment.py

The Python script takes six arguments:

--experiment_path: The path to the experiment-defining JSON file.
--subscription_id: Subscription ID that the experiment should be created under.
--resource_group: The resource group where the experiment should be created.
--name: The name of the experiment.
--log_path: Path to a file where logs shall be written. (Optional)
--log_level: Integer that determines what logs should be printed to stdout:

0: print everything
1: print status + errors only
2: print fatal errors only

Once invoked, the Python script will read the JSON file pointed to by --experiment-path, and determine the resources and faults involved. It will then execute Azure CLI calls to:

1. Create user managed identity for Chaos Agent (if Agent-based faults are involved).
2. Install chaos mesh on Kubernetes Clusters (if AKS resources are involved).
3. Create appropriate Chaos Targets on the desired Azure resources.
4. Add the desired capabilities to the targets.
5. Install chaos agent on virtual machines (if Agent-based faults are involved)
6. Create the experiment.
7. Grant experiment access to the target resources.

If the Python script exits successfully, the experiment is ready to run. You may run the experiment through the Azure portal, or you may run it through CLI. The Bash script in the next section will invoke the python script, run the prepared experiment, and monitor its progress.

conduct_experiments.sh

The Bash script takes its arguments through environment variables, making it easy to use it through Azure DevOps Pipelines. All variable names are in all caps, because Azure DevOps Pipelines converts all YAML variables into fully capitalized environment variables. The following environment variables must be set:

SUBSCRIPTION_ID: Subscription ID for experiments and involved resources.

RG_NAME: Resource group for the experiments.

LOG_FILE_PATH: Path to a file where logs shall be written.

LOG_LEVEL: Log level input for the Python Script defined in the last section.

SETUP_EXPERIMENT_SCRIPT_PATH: Path to the Python script in the last section.

EXPERIMENT_DIR: Path to a directory containing experiment files, described below.

The Bash script is given a directory containing experiment files. The only required file is “experiment.json” which contains the experiment-defining JSON code described in the overview of this user manual. The directory may also contain the following optional files:

preexperiment.sh: Script to be run immediately before starting the experiment.

experiment.sh: Script to be run in parallel with the experiment and terminated after.

postexperiment.sh: Script to be run after the experiment is completed.

Once invoked, the Bash script will check the directory pointed to by EXPERIMENT_DIR. If this directory contains a file called “experiment.json”, it will perform the following actions:

1. Invoke the Python script with experiment.json to prepare the experiment.
2. Run preexperiment.sh if present.
3. Start experiment.sh in the background if present.
4. Start the experiment.
5. Wait until the experiment terminates.
6. Terminate the execution of experiment.sh if present via SIGTERM*.
7. Run postexperiment.sh if present.

If EXPERIMENT_DIR does not contain an experiment.json, these steps are executed on every sub-directory, making it possible to run multiple experiments in a single script invocation.

*The experiment.sh script must be able to terminate within 10 seconds of receiving a SIGTERM from the Bash script after the experiment.

Invocation Instructions

Direct Script Invocation

Both the Bash script and the Python script can be invoked directly from a shell. For the scripts to work, the shell must be able to execute Azure CLI commands. This requires the user to install Azure CLI, and then use it to authenticate with a valid Azure subscription with the “az login” command. There are multiple ways to authenticate an Azure CLI session, such as through an Azure Virtual Machine with a managed identity, service principle, or browser. As long as the session is authorized to create azure resources and role assignments for those resources, any method of authentication is fine. Once logged into an Azure CLI session in the shell, either script can be directly executed as long as they are provided with the correct variables and files. Keep in mind that the Python script receives its input through command line arguments, and the Bash script receives its input through environment variables.

Azure DevOps Pipeline Invocation

To invoke the framework with an Azure DevOps pipeline, use the Azure CLI task in your pipeline YAML. The Azure CLI task will authenticate with the Azure Resource Manager using an Azure DevOps service connection. Create the service connection in the Azure DevOps project settings, and use the service connection’s name as the “azureSubscription” input for the task.

An example of a full pipeline.yaml file that utilizes this framework is located in the "pipeline.yaml" file included with this user manual. An example of just the Azure CLI task is below:

```
- task: AzureCLI@2

inputs:

azureSubscription: serviceconnectionname

scriptType: 'bash'

addSpnToEnvironment: true

scriptLocation: 'scriptPath'

scriptPath: '$(Build.Repository.LocalPath)/tools/conduct_experiments.sh'

displayName: 'Experimentation'
```

Examples

Load Balanced VM Environment

The following examples use an Azure environment with two virtual machines connected to a load balancer. The VMs run web servers, and incoming requests are routed from the load balancer to either of the servers. To provision this environment, the Terragrunt files located in the "virtual-machine-env/resources" directory included with this user manual should be used. Before invoking Terragrunt, set the following environment variables:

SUBSCRIPTION_ID: ID of Azure subscription to be used for these resources.

RG_NAME: Resource group to provision the environment in.

LOCATION: Location ID for the environment. Make this the same location as the resource group.

PUBLIC_KEY: Public key for VM access.

KV_NAME: Name for the Key Vault.

VAULT_NAME: Name for the key vault.

LB_NAME: Name for the load balancer.

NSG_NAME: Name for the VM subnet network security group.

VM1_NAME: Name for the first VM.

VM2_NAME: Name for the second VM.

VNET_NAME: Name for the virtual network.

VM1_DNS_LABEL: Name for the DNS label for the first VM's public IP.

VM2_DNS_LABEL: Name for the DNS label for the second VM's public IP.

VM_IMAGE_ID: Resource ID for a web-server VM image to be used for the VMs.

Once these variables are set, run "terragrunt run-all apply" in the "virtual-machine-env/resources" included with this user manual, and the resources will be provisioned.

VM Shutdown Scenario

This experiment will shut down one VM, wait some time, then shut down the other. The files for this example are located in the "virtual-machine-env/experiments/experiment1" directory included with this user manual, but require the user to replace SUBSCRIPTION_ID, RESOURCE_GROUP, LOCATION_ID, VM1_NAME, and VM2_NAME with the right values in the experiment.json file. When the placeholder values have been replaced with the correct values for the desired environment, invoke the framework with the experiment directory set to the full path of the "virtual-machine-env/experiments/experiment1" directory via input variables. The included experiment will be parsed, and the following steps, and included Azure CLI commands, will be executed by the framework:

1. Create Chaos Targets on the virtual machines.
2. Add the VM shutdown capability to the targets.
3. Create the experiment.
4. Grant experiment access to the target resources.
5. Wait until servers are active (preexperiment.sh)
6. Run the experiment

CPU Pressure Scenario

This experiment will use the stress-ng library to cause virtual CPU pressure on one of the VMs. The files for this example are located in the "virtual-machine-env/experiments/experiment2" directory included with this user manual, but require the user to replace SUBSCRIPTION_ID, RESOURCE_GROUP, LOCATION_ID, VM1_NAME, and VM2_NAME with the correct values in the experiment.json file. When the placeholder values have been replaced with the right values for the desired environment, invoke the framework by pointing it to the "virtual-machine-env/experiments/experiment1" directory via input variables. The included experiment will be parsed, and the following steps, and included Azure CLI commands, will be executed by the framework:

1. Create user managed identity for Chaos Agent.
2. Create Chaos Targets on the VMs.
3. Add the CPU pressure capability to the targets.
4. Install the chaos agent on the virtual machines.
5. Create the experiment.
6. Grant experiment access to the virtual machines.

Key Vault Access Denial

This experiment will temporarily block access to the key vault, simulating the effects of a permissions mishap or authentication failure. The files for this example are located in the "virtual-machine-env/experiments/experiment3" directory included with this user manual, but require the user to replace SUBSCRIPTION_ID, LOCATION_ID, RESOURCE_GROUP, and VAULT_NAME with the correct values in the experiment.json file. When the placeholder values have been replaced with the correct values for the desired environment, invoke the framework by pointing it to the "virtual-machine-env/experiments/experiment3" directory via input variables. The included experiment will be parsed, and the following steps, and included Azure CLI commands, will be executed by the framework:

1. Create Chaos Target on the key vault.
2. Add the Deny Access capability to the target.
3. Create the experiment.
4. Grant experiment access to the key vault.
5. Run the experiment.

Azure Kubernetes Service Cluster Environment

The following example uses an Azure Kubernetes Service Cluster. The cluster runs a single web server pod. To provision this environment, the terragrunt files located in the "aks-env/resources" directory included with this user manual should be used. Before invoking Terragrunt, set the following environment variables:

SUBSCRIPTION_ID: ID of Azure subscription to be used for these resources.

RG_NAME: Resource group to provision the environment in.

CLUSTER_NAME: The name of the AKS cluster.

Once these variables are set, run “terragrunt run-all apply” in the “aks-env/resources” directory included with this user manual, and the resources will be provisioned.

The preexperiment.sh script included in the following experiment will create the web server pod.

Pod Failure Scenario

This experiment will disable a pod under the “default” namespace within the cluster. The files for this example are located in the “aks-env/experiments/experiment1” directory included with this user manual, but require the user to replace SUBSCRIPTION_ID, RESOURCE_GROUP, LOCATION_ID, and CLUSTER_NAME with the correct values in the experiment.json file. The preexperiment.sh script for this experiment requires KubeCTL to be installed, so that it can create the web server pod in the cluster. When the placeholder values have been replaced with the correct values for the desired environment and KubeCTL has been installed, invoke the framework by pointing it to the “aks-env/experiments/experiment1” directory via input variables. The included experiment will be parsed, and the following steps, and included Azure CLI commands, will be executed by the framework:

1. Install chaos mesh on the Kubernetes Cluster.
2. Create Chaos Targets on the cluster.
3. Add the Pod Chaos capability to the target.
4. Create the experiment.
5. Grant experiment access to the cluster.
6. Run the experiment.

Maintenance and Future Improvements

There are many improvements that can be made to this framework. This section will detail the modifications and additions that should be made in order to ensure that these scripts remain useful.

Maintaining the JSON Interpretation Data

Before utilizing this section, one must understand two concepts within the experiment-defining ARM template that the Azure devs have decided to use. “Actions” are strings that refer to a specific Chaos Studio fault that appear in the definition of a particular step in the JSON. These actions have a corresponding “Capability ID” that must be used when enabling the fault on a target via Azure CLI.

As you add more faults to your experiments, you must ensure that those faults are supported in this action_to_capabilities global dictionary in the Python script. There must be a key-value pair, where the key is the action and the value is the capability. The correct strings for the action and capability can be found in the “Chaos Studio fault and action library”, currently located at https://learn.microsoft.com/en-us/azure/chaos-studio/chaos-studio-fault-library. The action strings are the “Urn” fields in the tables for each fault, and the capability IDs are the “Capability Name” fields in the same table.

Removing the Need to Maintain JSON Interpretation Data

Maintaining the necessary key-value pairs according to new/updated faults is not necessary if the framework automatically queries the Microsoft fault and action library tables mentioned above. The framework could be modified to use these online tables directly rather than keeping its own hard-coded data structure, and no maintenance would be needed. The only downside is that an internet connection would always be required, and the rare case of a mistake in Microsoft's tables.

Improving Data Collection Script Pipeline Output

The framework supports automatically running scripts (called experiment.sh) in parallel with running experiments. These scripts are run in the background and are not displayed on the pipeline’s standard output. Therefore, it is up to the user to ensure that the collected data is written to the pipeline's artifact staging directory and then published as artifacts. Improvements can certainly be made to this system so that the output of these scripts can be viewed in the standard pipeline output or even run in pipeline parallel jobs.

Automatic Stress-ng Installation

Some agent-based capabilities require the stress-ng application to be installed on target Linux VMs. The framework does not handle or verify the installation of stress-ng for these capabilities (specified in the Microsoft fault library). The framework can certainly be modified to detect when stress-ng is needed and handle the installation.

Windows Chaos Agent Support

Currently, the framework only supports Linux VMs for agent-based experiments. Adding windows support should not be too difficult. The framework just needs to recognize that a VM is a windows machine, so it doesn't execute the command to install the Linux chaos agent.

Input Checking

The scripts have limited verification of their input variables. For example, Azure naming limitations are not taken into account when the experiment name is provided to the Python script, potentially causing a runtime error if an invalid name is requested.

Limiting Experiment Permissions

Currently, the framework grants the experiment access to the target resources via the "Contributor" role (and also the "Reader" role for agent-based targets), regardless of the faults being used in the experiment. These privileges are sometimes higher than necessary. For example, there is a specific role that only grants VM shutdown/start permissions, and this is all an experiment needs if it only utilizes the VM shutdown capability. Modifications can be made to the framework to only grant the least permissions necessary to an experiment based on the capabilities being used.

Replacing Experiment Defining JSON

While the framework interpreting the same language as the Azure Resource Manager has its benefits, the ARM templates for experiments are likely to change due to the "preview" nature of Chaos Studio. It may be more practical to define a framework-specific experiment definition syntax, to make it easy for devs to write experiments and for framework maintainers to handle the input.

Automatically Generating Experiments

Since many Azure resources have a set of faults that Azure Chaos Studio supports, the framework can analyze a resource group and generate random experiments based on the resources present. This means that dozens of experiments could be automatically generated and run, possibly discovering vulnerabilities not seen by manually written experiments.