Integrating Security Practices into Backend Development and DevOps Environments

Romelo Gilbert

CYSE 368 / Internship

Watsco Inc

Instructor: Professor Teresa Duvall

Term: Spring 2026

Date: April 26, 2026

Table of Contents

1. Introduction

I decided to complete my internship at Watsco Inc because the position offered direct exposure to the type of modern backend development environment I want to continue working in professionally. My long term goal is to grow as a backend engineer with strong knowledge of secure software development, cloud infrastructure and DevOps practices. Watsco's technical environment gave me an opportunity to work with technologies that are highly valued in the software development field. Including JavaScript, TypeScript, SQL, Docker, Kubernetes, AWS, Helm, Jenkins, Bitbucket and monitoring tools such as Datadog. I was especially interested in the internship because it was not limited to a classroom simulation or isolated coding exercises. The work involved real systems, real production constraints and real business consequences when services, data flows or configurations did not behave as expected.

Watsco Inc is a large HVAC/R distribution company headquartered in Miami, Florida. HVAC/R refers to heating, ventilation, air conditioning and refrigeration. In practical terms, Watsco's business supports the movement of equipment, parts and supplies that contractors need to install, repair and maintain residential and commercial climate control systems. The company sells and distributes products such as air conditioning systems, heating equipment, heat pumps, refrigeration equipment, thermostats, motors, coils, compressors, filters and other replacement parts. These products are essential to homeowners, businesses, schools, warehouses, medical facilities, restaurants and many other organizations that depend on reliable indoor climate control and refrigeration. Because HVAC/R equipment is both technical and time sensitive, contractors depend on accurate inventory information, efficient ordering systems and reliable fulfillment processes.

A major part of Watsco's business is not only the physical distribution of equipment but also the digital infrastructure that supports that distribution. Contractors and business units rely on software systems to search products, check availability, place orders, manage fulfillment, update inventory and support field operations. The backend systems supporting these functions must be reliable, secure and scalable because errors can directly affect contractors, customers, warehouses, distribution centers and internal teams. This made the internship especially meaningful to me. I was not simply writing code for a small practice application. I was working in an environment where software connects directly to business operations.

At the beginning of the internship, I was introduced to a complex microservices environment. Multiple backend services communicate with each other through APIs and those services are deployed across development, staging and production environments. The development workflow includes source control in Bitbucket, pull requests for code review, Jenkins for build and deployment processes, Docker image builds, Kubernetes clusters, Helm managed configuration and monitoring through Datadog. My initial impression was that the system was large, layered and sometimes overwhelming. However, the structure also showed me how mature backend systems are organized in a professional environment. I quickly realized that success would require more than knowing how to write code. I needed to understand how code moved through the pipeline, how environments differed, how services authenticated to one another, how logs and metrics reflected system health and how configuration changes could affect the entire deployment process.

The primary learning objectives for my internship were established in my Memorandum of Agreement. These objectives included:

- Understanding and implementing CI/CD security practices
- Conducting security audits and risk assessments
- Automating security monitoring and incident response workflows
- Collaborating on secure Infrastructure as Code practices

Even though my job title was Backend Software Engineer, these objectives gave my internship a cybersecurity focus. Throughout the experience, I began to understand that cybersecurity is not separate from software development. It is built into testing, code review, dependency management, database access, authentication, deployment configuration, monitoring, incident response and operational decision making.

This paper describes my internship experience at Watsco Inc, including the organizational environment, my major duties and projects, the cybersecurity knowledge I applied, the ways ODU prepared me and the areas where the internship exposed gaps between academic learning and industry practice. It also evaluates how well the internship fulfilled my learning objectives and reflects on the most motivating, discouraging and challenging aspects of the experience. Overall, the internship helped me develop a more practical understanding of how backend development, cloud infrastructure, DevOps and cybersecurity intersect in a modern business environment.

2. Management Environment

The management environment at Watsco Inc was collaborative, practical and built around independent responsibility. Because I worked remotely, most communication occurred through Microsoft Teams chat and meetings. This created a work environment where clear communication was important. Unlike an in person office setting where it may be easier to ask a question to a team member sitting in the cubicle next to you. Remote work requires developers to explain issues clearly in writing, provide context, share screenshots or logs when needed and be proactive about asking for help. This helped me become more intentional in the way I communicated technical problems.

The team structure included several roles that interacted regularly with my work. The Conduit backend team included four developers, including a lead developer and was supported by a scrum master. The team also interacted frequently with a Conduit DevOps team member, the Order Fulfillment Team Lead and the Contractor Assist App Team Leader. There were also other business and technical teams involved depending on the feature, issue or deployment. This structure helped me see how backend development is rarely isolated. Even a seemingly small API change can involve business logic, database behavior or web application consumers.

Supervision within the internship was not overly rigid. The expectation was that developers actively seek to be productive, take ownership of assigned tickets and reach out when they encounter blockers. This style of supervision was effective for my development because it required me to think independently. Instead of waiting for detailed instructions for every step, I had to read tickets, understand the context, inspect the codebase, reproduce issues, investigate possible causes and propose solutions. When I got stuck, the team was supportive, but there was still an expectation that I would first make a serious attempt to understand the problem.

The open door culture was one of the strongest aspects of the management environment. Although the team expected independence, it was not an environment where asking questions

was discouraged. When I needed help understanding a service, a business workflow, a deployment issue or an authentication problem, I could reach out to teammates. This balance between independence and support helped me grow. It prevented me from relying too heavily on others, but it also gave me enough support to learn from more experienced engineers.

The management structure was also effective because it exposed me to cross functional decision making. Backend developers often had to coordinate with DevOps, business units or application teams. This helped me understand that software development in a business environment includes technical and organizational dependencies. For example, when working on deployment issues, it was not enough to know that code passed tests. The correct environment variables had to be present, network access had to be configured, IP addresses sometimes had to be whitelisted and business users had to validate that the behavior matched operational expectations. This taught me that technical work is successful only when it works within the larger organizational workflow.

One limitation I observed was the lack of a clear pipeline for lateral movement into DevOps. I was interested in gaining deeper experience with DevOps responsibilities because my learning objectives were strongly related to CI/CD security, infrastructure, monitoring and incident response. However, access to certain tools and privileges was limited and there did not appear to be a formal pathway for developers to temporarily rotate into DevOps responsibilities. This was somewhat discouraging because I could see how DevOps work connected closely to my cybersecurity objectives, but there were practical barriers to gaining hands on experience in that area. Still, the limitation also helped me think strategically about how to earn opportunities. Rather than expecting access automatically, I began thinking about how I could demonstrate value by solving problems, building useful tools or improving workflows that support DevOps objectives.

3. Major Work Duties, Assignments and Projects

My major work duties during the internship centered on backend development, API testing, SQL query work, dependency vulnerability remediation, infrastructure related configuration, monitoring and deployment support. While these tasks may appear different on the surface, they were connected by a common theme. Each one supported the reliability, security and maintainability of business systems. The experience helped me understand that backend engineering is not limited to writing endpoints. It includes ensuring that services process data correctly, authenticate requests properly, fail safely, deploy consistently and provide enough visibility for teams to detect problems.

One of my recurring responsibilities was developing and testing APIs. I worked on services used by internal systems and business units, including APIs connected to order fulfillment and contractor facing workflows. Writing tests for a key internal API required me to think carefully about input scenarios, acceptable parameters, edge cases and expected outputs. Before this experience, I sometimes thought about testing mainly as a way to confirm that code works. During the internship, my view matured. I began to see testing as a way to prevent regressions, protect assumptions, document expected behavior and reduce risk before code reaches production. In a distributed backend environment, one poorly handled input can create downstream issues for other services or clients.

Jest testing was especially important because some of the application logic depended on multiple services and dependencies. Writing tests required mocking external calls, stubbing responses and verifying behavior under different conditions. This improved my understanding of how modular code should be structured. Code that is difficult to test is often code that is tightly coupled or unclear in responsibility. As a result, testing pushed me toward better design habits, including separating responsibilities, using services consistently and considering how changes might affect consumers.

I also worked on refactoring SQL queries for order fulfillment workflows. One example involved a scenario where multiple rows could exist for the same item number and the system needed to aggregate quantity values correctly. This task showed me how important data accuracy is to business operations. If quantity values are not calculated correctly, the problem is not simply technical. It can affect inventory, ordering, fulfillment and decision making. From a cybersecurity perspective, the task also reinforced the importance of secure database interaction. Any time software constructs queries based on user or service inputs, developers must consider whether those inputs are validated, sanitized and handled safely. A correct query is not enough if the query building process introduces risk.

Another important task involved resolving a high severity vulnerability in the fast-xml-parser dependency. This was one of the clearest examples of cybersecurity intersecting with everyday development work. The vulnerable code was not necessarily code that I or my team wrote directly. It existed in a third-party library that the application depended on. This showed me that modern software security extends beyond the internal codebase. Every dependency can become part of the attack surface. Updating the package to a patched version helped me understand the importance of dependency scanning, vulnerability tracking and timely remediation. It also made me more aware of how teams must balance security fixes with compatibility testing because dependency upgrades can sometimes introduce breaking changes.

I worked with monitoring and alerting using Datadog and PagerDuty. This included configuring dashboards and alert thresholds based on application logs and error patterns. For example, alert conditions were created for repeated 5XX errors, high volumes of 404 responses and rapid increases in 4XX responses. This work gave me practical exposure to operational monitoring. In development, it is easy to focus on preventing bugs before deployment. Monitoring taught me that production systems also require continuous observation after deployment. Even well tested systems can encounter unexpected input, external service failures, resource issues, network problems or configuration mistakes. Monitoring provides visibility into those conditions.

I also worked on tasks related to the CSD (Cloud Suite Distribution) post cycle count process. This feature supported a workflow where distribution center workers could perform inventory cycle counts using handheld devices and barcode scanning rather than relying only on manual handwritten processes. The system supported retrieval and update operations so that inventory data could be reflected in the database. This task was important because it connected backend software to physical warehouse operations. A failure in the backend could affect the accuracy of inventory counts. Inaccurate inventory can affect ordering, fulfillment and business planning. It also showed me how backend developers must understand the real world workflow their code supports.

Another project involved implementing an external product search feature for requests from a business unit. This work occurred during a broader transition from a V1 JavaScript codebase to a V2 TypeScript codebase. I was able to refactor and restructure logic while building the new endpoint. Creating a new service that extended an existing service improved code reuse and helped avoid duplicating logic. This experience reinforced the value of maintainability. In a large codebase, repeated logic can create security and reliability problems because fixes must be applied in multiple places. Reusable, structured code reduces the chance that one path is updated while another remains vulnerable or incorrect.

The transition from V1 to V2 backend services became especially important during go-live testing. During this phase, the business unit executed tests across services and endpoints to validate that responses matched expected behavior. When issues were discovered, I helped identify and correct incorrectly implemented endpoints. This work required understanding the modular design of the codebase and tracing issues quickly. The experience reinforced the importance of maintainable architecture. A system that is well organized is easier to debug during high pressure moments, which reduces downtime and helps teams respond effectively during production transitions.

Deployment support was another major part of my learning. During production deployment of new cycle count functionality, the team encountered database errors, missing environment variables and IP address whitelisting requirements. These issues showed me that deployments are not only about whether the code compiles. Production readiness also depends on configuration, network access, secrets, database permissions, environment and coordination across teams. A single missing environment variable can cause a feature to fail even if the code is correct. This experience helped me understand why deployment checklists, automated validation and configuration management are important parts of secure and reliable CI/CD processes.

Local development troubleshooting also became an important learning experience. In one case, multiple issues were contributing to a problem. Outdated environment variable values, expired VPN credentials and an IP whitelisting requirement. This taught me to isolate variables methodically. Troubleshooting complex systems requires patience and discipline because multiple overlapping problems can make the true root cause difficult to identify. From a security perspective, the issue also showed how authentication, VPN access and network controls work together. Access to a system depends on more than having the correct code. It also depends on identity, credentials, network location and approved communication paths.

Overall, my work duties were necessary to the business because they supported system reliability, data accuracy, secure access, operational visibility and deployments. Watsco's digital systems support contractors, warehouse workers, business units and customer facing applications. The tasks I completed contributed to those systems by improving tests, correcting endpoints, securing dependencies, monitoring, managing infrastructure configuration and supporting production readiness.

Table 1. Internship Duties and Business Value

Duty or Project	Primary Technical Focus	Business Value
Jest API testing	Validation, edge cases, regression prevention	Improves reliability of services used by internal systems and business consumers.

SQL query refactoring	Data accuracy, aggregation, secure query behavior	Supports accurate order fulfillment and protects database interactions.
Dependency vulnerability remediation	Patch management and third-party risk	Reduces risk from vulnerable libraries and improves software supply chain security.
Datadog/PagerDuty alerting	Monitoring, thresholds, automated notification	Helps teams detect abnormal behavior and respond faster.
Helm/Kubernetes configuration	Infrastructure as Code and environment management	Supports consistent deployment and secure configuration across environments.
Go-live support	Endpoint validation, troubleshooting, production readiness	Helps business units transition to new services with less disruption.

4. Application of Cybersecurity Skills and Knowledge

The internship changed my understanding of cybersecurity by showing me that security is not a single activity performed at the end of development. Instead, it is a mindset and set of practices that must be integrated throughout the entire software lifecycle. Before the internship, my cybersecurity knowledge was mostly academic. I understood concepts such as authentication, encryption, vulnerabilities and risk at a high level, but I had less experience applying those concepts in a production environment. On the job, I saw that cybersecurity decisions appear in ordinary development tasks. How inputs are validated, how dependencies are updated, how secrets are stored, how services authenticate.

One cybersecurity skill I applied was vulnerability awareness and patching. The fast-xml-parser vulnerability was an important example because it demonstrated how a dependency can introduce risk even when the application code itself appears stable. Modern applications rely heavily on open source libraries and third-party packages. This creates a software supply chain risk because the security of the application depends partly on the security of code maintained outside the organization. By updating a vulnerable package, I gained practical experience with remediation. I also learned that patching is not always as simple as changing a version number. Teams must ensure the patched version is compatible, tests still pass and the application behaves correctly after the update.

Another cybersecurity area was secure CI/CD. In the environment I observed, source code is stored in Bitbucket, changes are reviewed through pull requests, Jenkins is used for build and deployment activity, Docker images are built, tests run during the build process and deployments are made to Kubernetes with Helm managed configuration. Security enters this workflow in several ways. Pull requests provide peer review before code is merged. Automated tests help catch errors before deployment. Vulnerability scanning helps identify risky dependencies. Image builds create deployable artifacts that can be promoted across environments. Helm configuration controls how applications run inside Kubernetes. Each step can either reduce risk or introduce risk depending on how well it is managed.

Authentication and authorization were also central to my learning. I worked with request flows that required the correct JWT token type. In the environment, different token types may be used for different contexts. Including user tokens, machine-to-machine tokens, app tokens and web app tokens. Using the wrong token can result in 403 authorization errors. This experience

helped me understand that authentication is not simply about whether a token exists. It is about whether the token represents the correct identity, audience, scope and context for the request. In a microservices environment, services must be able to verify not only that a request is authenticated, but also that it is authorized for the specific operation.

Network security became more practical to me through VPN and IP whitelisting issues. When my local development application could not communicate with a business unit's network servers, the problem eventually involved approved network access. This showed me that cybersecurity often works through layers. A valid application configuration may still fail if the network path is not approved. A valid user may still be blocked without VPN connectivity. A service may still be rejected if the request originates from an unapproved IP address. These controls can be frustrating during troubleshooting, but they exist to reduce unauthorized access and limit exposure.

Secrets management was another area where my understanding grew. In production, sensitive values are stored in AWS Secrets Manager rather than directly in the codebase. This is important because credentials, API keys and other secrets should not be exposed through repositories or static configuration files. However, I also observed that development and staging environments handled some variables differently, including storing values in a private Bitbucket repository. This raised important questions about acceptable risk. Development and staging systems often require convenience and speed, but they can still contain sensitive configuration or access paths. The contrast between non-production and production environments helped me understand that security decisions often involve balancing risk, cost, operational convenience and business needs.

Database security became relevant through SQL query work. When constructing or refactoring queries, developers must be careful with input handling. SQL injection is a known vulnerability, but the internship helped me connect that concept to real work. A query written to retrieve or aggregate business data can become a security concern if user input is inserted unsafely. Even when using query builders or parameterization, developers must understand how data is passed into the query and whether assumptions about types, values or formats are valid. I learned that database work requires both correctness and caution.

Monitoring and alerting gave me a broader view of security. Before this internship, I often thought about security mainly as prevention. In practice, prevention is only one part of the picture. Systems also need detection and response. Datadog dashboards and PagerDuty alerts help teams identify abnormal behavior, such as increased 5XX errors or spikes in 4XX responses. A 5XX error spike may indicate a system failure, unhandled exception, dependency outage or resource issue. A 4XX spike may suggest unusual client behavior, authorization problems, misuse of endpoints or a breaking change. These signals may not always represent a security incident, but they are useful for risk awareness and operational response.

Overall, my on-the-job experience changed my understanding of cybersecurity from a set of abstract topics into a practical engineering discipline. I learned that cybersecurity is embedded in code quality, dependency management, authentication, database practices, network controls, secrets management, infrastructure configuration, monitoring and deployment workflows. This experience made me more aware that a backend developer has security responsibilities even when the job title is not security engineer. Every code change, configuration update or deployment decision can affect the security posture of the system.

5. ODU Curriculum Preparation and Gaps

The ODU cybersecurity curriculum helped prepare me for the internship by giving me a foundation in core security concepts. Courses related to networking, system security, risk and secure computing helped me understand why authentication, access control, encryption, monitoring and vulnerability management matter. Even when the internship required tools I had not used in class, the underlying concepts were familiar enough that I could connect them to what I had learned academically.

One important connection was authentication and authorization. In coursework, authentication is often discussed as a process for verifying identity, while authorization determines what an authenticated user or system is allowed to access. During the internship, I saw this distinction in practice through JWT tokens and service to service communication. A token was not useful simply because it existed, it had to be the correct type of token for the correct request context. This reinforced the academic concept that access control must be specific and intentional.

Another connection was secure coding and input validation. ODU coursework emphasized that software vulnerabilities can arise when applications fail to validate or sanitize input. This became practical when I worked with API parameters, SQL queries and edge cases involving data structures. Testing APIs for different acceptable parameters helped me see how input assumptions can fail. Refactoring database queries helped me understand how data handling can become a security issue if values are not handled safely. The classroom concept became more meaningful because I saw how input validation affects real services and business workflows.

The curriculum also helped prepare me to think about risk. Risk assessment in class can sometimes feel formal or theoretical, but in the internship I saw how risk appears in everyday decisions. Manual deployments introduce the possibility of inconsistency or human error. Storing environment variables in a repository may be acceptable in certain non-production contexts but still introduces exposure risk. Missing environment variables can cause deployment failures. IP whitelisting can prevent unauthorized access but also create troubleshooting challenges. These examples helped me connect academic risk concepts to operational decisions.

At the same time, the internship revealed gaps between academic preparation and industry practice. One major gap was DevOps tooling. Before the internship, I had limited academic exposure to tools such as Kubernetes, Helm, Jenkins, Docker image pipelines and Datadog. These tools are central to modern software delivery, especially in cloud based and microservices environments. While ODU provided a strong conceptual foundation, the internship showed me that students also need practical exposure to deployment pipelines, container orchestration and observability platforms.

Another gap was the complexity of production systems. Academic assignments are often scoped so that one student can complete them independently. Real systems are different. They involve multiple services, legacy code, business rules, environment specific configuration, shared libraries, external dependencies, network restrictions and teams with different responsibilities. The internship taught me that real world engineering requires navigating ambiguity. Problems are not always clearly defined and the first error message is not always the root cause. This was especially clear when troubleshooting local environment issues.

The internship also exposed me to the importance of observability, which I had not fully appreciated from coursework. In class, students may learn about logs, alerts and incident response conceptually, but seeing Datadog dashboards in a production context made the concept more concrete. Metrics, logs and traces help developers understand system behavior. They also support security by making abnormal patterns visible. This is an area where practical labs could strongly supplement cybersecurity education.

Despite these gaps, ODU prepared me in an important way, it taught me how to think about systems through a security lens. Even when I did not know a specific tool, I could still ask the right types of questions. What data is being protected? Who is allowed to access it? How is identity verified? What happens if this input is unexpected? What could fail during deployment? How would the team detect a problem? These questions helped me connect my academic background to my internship responsibilities.

Overall, the ODU curriculum gave me a foundation, but the internship added context, complexity and hands on application. The combination was valuable. Coursework gave me the vocabulary and concepts, while the internship showed me how those concepts appear in a professional backend and DevOps environment. Moving forward, I believe cybersecurity students benefit most when academic theory is paired with practical exposure to cloud infrastructure, CI/CD pipelines, monitoring and real world troubleshooting.

6. Learning Objectives Assessment

The learning objectives in my Memorandum of Agreement provided a useful structure for evaluating the internship. The four objectives were: 1) understand and implement CI/CD security practices. 2) conduct security audits and risk assessments. 3) automate security monitoring and incident response. 4) collaborate on secure Infrastructure as Code. My experience addressed each objective, although some were fulfilled more directly than others because my role was primarily backend development rather than a dedicated DevOps or security role.

The first objective, understanding and implementing CI/CD security practices, was strongly supported. I gained practical exposure to a CI/CD workflow involving Bitbucket, pull requests, Jenkins, Docker builds, vulnerability scanning, Kubernetes deployments and Helm configuration. I learned that CI/CD security is not one tool or one step. It is a series of controls and practices throughout the pipeline.

This objective was fulfilled not because I built the entire pipeline from scratch, but because I observed and worked within it. I saw how my code changes moved through the process and how security related checks affected development. I also saw areas for improvement, such as manual deployment steps and configuration validation. This gave me a realistic understanding of CI/CD security. Organizations often have strong controls in place, but there are still opportunities to reduce human error and automate more checks.

The second objective, conducting security audits and risk assessments, was fulfilled indirectly but meaningfully. I did not perform a formal audit with a checklist, report and remediation plan. However, many of my tasks involved identifying risks and understanding system behavior. Resolving a vulnerable dependency required recognizing third-party software risk. Refactoring SQL queries required awareness of database security and input handling. Observing manual deployments raised concerns about human error. Troubleshooting missing environment variables and IP whitelisting issues showed how configuration and network controls

can affect system availability and access. Reviewing Datadog dashboards helped me compare expected system behavior with actual behavior.

This objective helped me understand that risk assessment begins with awareness. A formal audit is a structured process, but the mindset behind auditing can be developed through everyday engineering work. Developers can identify where systems are fragile, where assumptions exist, where sensitive data is stored, where access controls are enforced and where manual steps create risk. My internship helped me develop this mindset. I became more aware of how small technical decisions contribute to the overall security posture of a system.

The third objective, automating security monitoring and incident response, was one of the most directly fulfilled objectives. My work with Datadog and PagerDuty gave me hands on experience configuring alerts based on system behavior. By setting thresholds for 5XX errors, 404 errors and general 4XX errors, I learned how automated notifications can help teams respond quickly to abnormal conditions. This experience showed me that monitoring is not only about collecting data. It is about deciding which signals matter, what thresholds indicate concern and how alerts should be routed so the right people can respond.

This objective also expanded my understanding of incident response. I had previously thought of incident response mainly in terms of major security events, but the internship showed me that incident response can begin with operational signals. A sudden increase in server errors may indicate a failing service. A spike in client errors may indicate misuse, broken clients, authorization problems or unexpected traffic. Monitoring provides early warning. Automated alerting reduces the chance that teams will miss important signals. Even when an alert does not indicate a security incident, the process of detection, triage and response supports system resilience.

The fourth objective, collaborating on secure Infrastructure as Code, was supported through my work with Helm and Kubernetes configuration. Updating Helm charts to modify Kubernetes pod environment variables gave me direct exposure to how infrastructure and application configuration are managed as code. I learned that IaC is not only about creating infrastructure quickly. It is also about making configuration repeatable, reviewable and consistent. However, IaC can also introduce risk if sensitive values are handled incorrectly or if configuration differs too much across environments.

This objective was partially fulfilled because I worked with infrastructure configuration, but I did not fully own IaC security scanning or policy enforcement. Still, the experience was valuable because it showed me the security implications of infrastructure configuration. I saw how production secrets were handled differently from development and staging values. I saw how environment variables affect application behavior. I saw how configuration mistakes can cause deployment failures. These lessons helped me understand why secure IaC practices matter and why infrastructure changes should be reviewed with the same seriousness as application code changes.

Overall, the internship fulfilled the learning objectives well. Some objectives were achieved through direct hands on work, while others were achieved through observation, troubleshooting and participation in the development lifecycle. The experience gave me a stronger understanding of how cybersecurity principles operate in a real DevOps environment and helped me identify areas where I want to continue growing, especially CI/CD automation, cloud security and infrastructure security.

7. Motivating or Exciting Aspects of the Internship

The most motivating aspect of the internship was the opportunity to work with modern technologies in a real business environment. Tools such as Kubernetes, Docker, AWS, Jenkins, Helm, Datadog, JavaScript, TypeScript and SQL are widely used across the software industry. Working with these tools helped me feel that I was developing skills that would be valuable beyond the internship. It also gave me confidence that I can contribute in professional engineering environments that use complex cloud and microservices architectures.

Another exciting aspect was working in a microservices environment. Microservices can be difficult to understand at first because functionality is distributed across multiple services. However, this architecture also helped me see how large systems are broken into smaller pieces with specific responsibilities. I learned that backend development in this type of environment requires understanding not only the service I am working on, but also the services it depends on and the clients that depend on it. This made the work more challenging, but also more interesting.

Monitoring work was also exciting because it expanded my view beyond code. Reviewing Datadog dashboards and configuring alerts helped me understand how running systems communicate their health. Seeing logs, metrics and error patterns gave me a better appreciation for operational engineering. It made me more interested in DevOps and site reliability practices because those areas connect development, infrastructure and security in a very practical way.

The internship was motivating because it showed me a path forward professionally. I entered with backend development interests, but the experience strengthened my interest in secure backend engineering and DevOps related work. I enjoyed the parts of the internship that required me to connect application logic with deployment behavior, monitoring and infrastructure. This helped me recognize that I want to continue building skills at the intersection of backend development, cloud systems and cybersecurity.

8. Discouraging Aspects of the Internship

The most discouraging aspect of the internship was the lack of a clear pathway for lateral movement into DevOps. Because my learning objectives were strongly connected to CI/CD security, monitoring, incident response and Infrastructure as Code. I was interested in gaining deeper access to DevOps tools and responsibilities. However, there did not appear to be a formal process for a backend developer or intern to rotate into DevOps tasks, shadow DevOps work in a structured way or gradually gain additional permissions for learning purposes.

This was discouraging because I could see how much DevOps work connected to my academic and career goals. CI/CD pipelines, Kubernetes configuration, secrets management, infrastructure policy and monitoring tools all play a major role in secure software delivery. While I was able to observe and participate in some related tasks, there were still limits to what I could access or perform directly. Those limits are understandable from a security perspective because production infrastructure and deployment tools require controlled access. However, from a learning perspective, it created a barrier.

The complexity of access and permissions was also occasionally discouraging. When trying to troubleshoot certain issues or learn more about infrastructure behavior, limited access could slow down investigation. Again, this is understandable because organizations must protect

sensitive systems, but it can make learning more difficult. It also showed me that professional environments must balance security with productivity. Too much access creates risk, but too little access can slow down problem solving and limit growth.

Despite these discouraging aspects, they were also educational. They helped me understand organizational realities that are not always visible in coursework. Security, access control and role boundaries affect how people work. Career growth inside an organization may require more than interest. It may require demonstrating value, building trust and finding practical ways to contribute to adjacent teams. These lessons are important for my future professional development.

9. Challenging Aspects of the Internship

The most challenging aspect of the internship was the complexity of the system. The codebase, services, environments and deployment processes were all interconnected. Early in the internship, it was difficult to understand where certain logic lived, which service owned which responsibility and how changes moved through the pipeline. This was very different from smaller academic projects where the entire system can usually be understood by one person. In a professional microservices environment, understanding must be built gradually.

One challenge was tracing data flow across services. A request might enter through a gateway, pass authentication checks, reach a backend API, call another service, query a database and return data to a web or mobile client. When something goes wrong, the cause may exist at any point in that chain. It may be an input issue, an authorization issue, a database issue, a network issue, a configuration issue or a bug in application logic. Learning to troubleshoot in this environment required me to slow down and investigate systematically.

Another challenge was understanding environment differences. Development, staging and production environments may be designed to behave similarly, but they are not always identical. Different environment variables, secrets, network rules, data sets and permissions can produce different behavior. This became clear during deployment and troubleshooting work. A feature may work locally but fail in staging because of a missing environment variable. A service may work in one network context but fail when an IP address is not whitelisted. These differences forced me to think about configuration as part of the system, not as an afterthought.

Authentication issues were also challenging. Working with multiple JWT token types required understanding the correct context for each request. When I used the wrong token type and encountered 403 errors, I had to learn how authentication boundaries were enforced. This was frustrating at first, but it became a valuable lesson. In complex systems, access failures are not always bugs, sometimes they are security controls working correctly. The challenge is to understand why access is denied and whether the request should be allowed.

Overall, the challenges of the internship helped me grow. They forced me to become more patient, more analytical and more comfortable with ambiguity. I learned that complex systems cannot be understood all at once. They require repeated exposure, careful investigation and willingness to ask questions. I also learned that frustration is part of the learning process, especially when working in a mature production environment.

10. Recommendations for Future Interns

Future interns in this type of role should prepare by building a strong foundation in backend development. They should understand how APIs work, including HTTP methods, request parameters, response structures, status codes, authentication and error handling. Much of the work involves understanding how services communicate and how data moves between clients, APIs, databases and other services. Interns who are comfortable reading backend code and tracing request flows will adapt more quickly.

Future interns should also practice debugging web applications. Debugging is one of the most important skills in a complex environment. Interns should be comfortable reading logs, reproducing issues, isolating variables, checking configuration and forming hypotheses about root causes. They should understand that the first error message may not always identify the actual problem. A disciplined debugging process is more valuable than guessing.

A solid understanding of SQL is also important. Backend systems often depend heavily on databases, so interns should understand how to write and read queries. They should understand joins, filtering, aggregation, data types and basic security concerns such as SQL injection. Even if an organization uses query builders or ORMs, developers still need to understand what the database is doing and how data is being retrieved or modified.

Future interns should become familiar with Git and pull request workflows. Professional development depends on source control, code review, branching and collaboration. Interns should know how to create branches, commit changes clearly, resolve merge conflicts and respond to code review feedback. They should also understand that pull requests are not only administrative steps, they are important quality and security controls.

Interns should also learn basic DevOps concepts before beginning. They do not need to be experts, but familiarity with Docker, CI/CD pipelines, Kubernetes, environment variables, secrets and monitoring tools will help. Understanding what a container is, how applications are deployed and why configuration differs across environments will make the internship less overwhelming. Even a small personal project using Docker and a simple CI workflow would provide useful preparation.

Another recommendation is to become comfortable with ambiguity. In professional environments, tickets are not always perfectly detailed, documentation may be incomplete and the person who originally wrote a service may not be available. Interns must be willing to investigate, read code, ask clarifying questions and learn incrementally. Being uncomfortable at first does not mean failing, it is part of working in complex systems.

Finally, interns should also develop communication skills. Remote work makes communication especially important. When asking for help, interns should explain what they are trying to do, what they have already checked, what error they are seeing and what they think may be happening. This makes it easier for teammates to help and shows that the intern has made an effort to understand the issue.

11. Conclusion

My internship at Watsco Inc was a valuable professional and academic experience because it showed me how backend development, DevOps and cybersecurity intersect in a real world environment. I entered the internship with an interest in modern backend technologies and

a desire to strengthen my cybersecurity knowledge. Through hands on work, I gained experience with API development, testing, SQL query refactoring, dependency vulnerability remediation, monitoring and Kubernetes configuration. Each of these experiences helped me understand how software systems are built, secured, deployed and maintained at scale.

One of my main takeaways is that cybersecurity is not separate from development. Security appears in everyday engineering decisions. Writing tests helps prevent unexpected behavior. Refactoring queries requires secure input handling. Updating dependencies reduces software supply chain risk. Managing secrets protects sensitive data. Using the correct JWT token enforces authentication boundaries. Configuring alerts improves detection and response. Managing infrastructure configuration affects deployment security and system reliability. These lessons changed my view of cybersecurity from a specialized activity into an integrated engineering responsibility.

Another important takeaway is that reliability and security are closely connected. A system that is difficult to deploy, monitor or troubleshoot can create risk even if it does not contain an obvious vulnerability. Missing environment variables, network access issues, manual deployment steps and misconfigured services can all affect availability. This helped me understand why modern cybersecurity includes not only preventing attacks, but also building resilient systems that can be observed, maintained and recovered when issues occur.

The internship will influence the remainder of my time at ODU by encouraging me to focus on practical, hands on learning. I want to continue building projects that involve APIs, containers, CI/CD pipelines, cloud services, monitoring and security controls. I also want to connect my coursework more intentionally to real world systems. When learning about risk, networks, authentication or secure coding, I now have practical examples from my internship that make those concepts more meaningful.

Professionally, the internship reinforced my interest in backend development while also increasing my interest in DevOps and cloud security. I still see backend engineering as a strong career path, but I now understand that the strongest backend engineers are not limited to writing business logic. They understand deployment, infrastructure, monitoring, security and operations. My goal moving forward is to continue developing as an engineer who can build reliable APIs while also understanding the systems and security controls that support them.

Overall, the internship helped bridge the gap between academic cybersecurity knowledge and professional software engineering practice. It gave me a clearer understanding of how complex systems operate and how security must be considered throughout the software lifecycle. Although the experience included challenges and some limitations, it strengthened my technical confidence, improved my problem solving approach and clarified the direction I want to take in my career.