I. M.

DevOps Engineer

5+ years of experience

Core competencies

DevOps Engineer with over 5 years of hands-on experience in architecting, deploying, and maintaining large-scale, resilient systems. Proficient in diverse technologies, specializing in containerization (Docker, Kubernetes), cloud platforms (AWS, GCP), automation scripting (Python, Bash), and CI/CD pipelines (GitHub Actions, GitLab CI). Expertise extends to infrastructure as code tools such as Ansible and Terraform. Well-versed in application development, Linux/Unix administration, networking, and security protocols. Renowned for problem-solving prowess and adaptability in fast-paced environments. Demonstrated ability to deliver projects within budget and schedule, prioritizing reliability and performance. Effective leader fostering teamwork and mentoring to optimize team productivity.

Technical capabilities

Cloud	**AWS** (5+ years), **GCP** (3+ year),
Containerization	**Kubernetes** (5+ years), **Docker** (5+years)
CI/CD	**GitHub Actions**/**GitlabCI** (4+ year), **Jenkins** (3+ year)
IaC	**Terraform** (5+ year), **CloudFormation** (2+ years)
Configuration Management	**Ansible** (5+years)
Monitoring	**Prometheus**/**Grafana** (3+ year), CloudWatch (3+year)
Database	**PostgreSQL**, **MySQL**(4+ years), **MongoDB**, **DynamoDB**(4+ years)
Scripting & Automation	**Python**(4+ years), **Bash**(3+ years)
Operating Systems	**RHEL/CentOS, Ubuntu, Amazon Linux**

Certifications

AWS Certified Solutions Architect
HashiCorp Certified: Terraform Associate

Education

The Kyrgyz State Technical University **Bhd. of Computer Science**

English: Upper Intermediate

Selected Projects

Real-time Log Analysis System with Elasticsearch

Role DevOps Engineer
Technologies AWS (EKS, ECS, ECR), Terraform, Helm, Kubernetes, Nginx Ingress Controller, GitHub Actions

Description:

Implemented a Real-time Log Analysis System with Elasticsearch to revolutionize our organization's monitoring and troubleshooting capabilities. This project aimed to provide a robust and scalable solution for aggregating, analyzing, and visualizing log data in real-time. As the DevOps engineer leading this initiative, I played a pivotal role in every phase of the project lifecycle, from inception to deployment and ongoing optimization.

The project began with a comprehensive analysis of our existing logging infrastructure and requirements gathering sessions with stakeholders to identify key pain points and desired outcomes. Following this, I architected a solution leveraging the Elastic Stack, with Elasticsearch as the core component for indexing and searching log data, Logstash for data ingestion and transformation, and Kibana for visualization and monitoring.

One of the primary challenges encountered was ensuring the scalability and resilience of the Elasticsearch cluster to handle the influx of log data from diverse sources while maintaining optimal performance. To address this, I designed and implemented a Dockerized deployment of Elasticsearch on a Kubernetes cluster, leveraging Kubernetes StatefulSets for managing persistent storage and dynamic scaling of Elasticsearch nodes based on resource utilization.
Another significant challenge was optimizing the Elasticsearch index configuration and shard allocation to balance query performance with storage efficiency. Through meticulous performance testing and benchmarking, I fine-tuned the index mappings, shard settings, and caching strategies to achieve optimal query response times and minimize resource overhead.

Furthermore, integrating real-time alerting and monitoring capabilities using Elasticsearch Watcher presented technical complexities in defining and configuring custom alert conditions. I overcame this challenge by collaborating with the development team to define actionable alert criteria and implementing Watcher alerts for critical system events, performance anomalies, and security incidents.

Responsibilities:

- Architected and deployed a scalable infrastructure using Docker and Kubernetes for orchestrating Elasticsearch clusters, ensuring high availability and fault tolerance.
- Developed Logstash configurations to ingest and preprocess log data from diverse sources, including application servers, network devices, and security appliances.
- Defined index templates, mappings, and lifecycle policies to optimize storage utilization and ensure efficient data retention.
- Utilized Elasticsearch Query DSL to build complex search queries and aggregations for real-time log analysis. Developed custom visualizations and dashboards in Kibana to provide actionable insights into system performance and error trends.
- Integrated Elasticsearch Watcher to set up proactive alerts for detecting critical events and anomalies in log data. Implemented centralized logging and monitoring using Beats agents for system metrics and cluster health monitoring.
- Fine-tuned Elasticsearch JVM heap settings, shard allocation strategies, and caching mechanisms to optimize query response times and reduce resource utilization. Conducted load testing and performance profiling to identify bottlenecks and scalability limits.
- Documented the project architecture, deployment procedures, and best practices for maintaining the Elasticsearch infrastructure. Provided training and technical support to development and operations teams on utilizing Elasticsearch for log analysis and troubleshooting.

Migration of LAPP Stack Application to WHM and cPanel

Role DevOps Engineer

Technologies AWS (EC2, Route53), WHM, cPanel, Atlassian Cloud, PostgreSQL, phpMyAdmin, Linux, PHP.

Description:

Migrated a LAPP stack application (Linux, Apache, PostgreSQL, PHP) from an AWS EC2 instance to a WHM/cPanel server hosted on Atlassian Cloud. This migration required careful planning and execution to ensure a smooth transition of both the website and PostgreSQL database to the new hosting environment. Throughout the process, I focused on minimizing downtime, optimizing data transfer, and ensuring seamless operation post-migration, ensuring that the application would be fully functional on the new infrastructure without disruption.

The project began by setting up the WHM/cPanel environment from scratch, including configuring necessary system settings and provisioning resources to meet the performance and scalability needs of the migrated application. After this, I migrated the website files to the new server, ensuring the correct directory structure and file permissions were in place for optimal security and performance. I also managed the PostgreSQL database migration, utilizing phpMyAdmin to facilitate database user creation, data export/import, and database management tasks, ensuring all data and configurations were properly transferred.

Key challenges included the installation and configuration of necessary PHP extensions to ensure compatibility with the application's specific requirements after migration. One of the primary obstacles was domain management, as cPanel's UI

only allowed setting up subdomains (e.g., `dev.site.com`) pointing to the `public_html` directory. However, I needed the main domain (`site.com`) to function as the root of the site. To overcome this limitation, I manually configured the domain settings through WHM's terminal, bypassing the restrictions of the cPanel interface, which ensured the website was accessible through the correct domain and directory.

Responsibilities:

- Migrated the LAPP stack (Linux, Apache, PostgreSQL, PHP) from AWS EC2 to WHM/cPanel on Atlassian Cloud.
- Copied all application code to the Atlassian Cloud server and configured necessary permissions for directories and files to ensure proper access and security.
- Managed PostgreSQL database migration, including data export, import, and user creation using phpMyAdmin.
- Installed and configured necessary PHP extensions for application compatibility.
- Configured website files in `public_html` and set up domain and SSL certificates for secure website access.
- Handled manual domain configuration in WHM to point the root domain (`site.com`) to the correct web directory.
- Provided post-migration support and documentation to ensure smooth operation on the new server infrastructure.

Seamless Migration of PostgreSQL Standalone Instance to Aurora (Amazon RDS) with Zero Downtime

Role DevOps Engineer
Technologies AWS (DMS, RDS, Aurora, PostgreSQL, VPC SG), Standalone Postgresql server

Description:

Successfully migrated the PostgreSQL standalone instance to Amazon Aurora as a PostgreSQL-Compatible global database, an Amazon Relational Database Service (RDS) solution, to meet the evolving needs of an IoT company. The decision to transition from a standalone setup to a cloud-native infrastructure was driven by the necessity for scalability, fault tolerance, and simplified management. We used a global database to accommodate users from different regions as the company expanded. This setup provides low latency access to data for users in different geographic locations, improving the overall user experience and supporting the company's growth trajectory.

The migration process began with meticulous configuration of the source PostgreSQL database server to enable remote connections and replication. A new DB user named 'repuser' was created with superuser privileges, ensuring seamless integration with the migration toolset. Subsequently, an Aurora cluster was provisioned within the AWS RDS environment, providing a robust foundation for the upcoming data migration. The entire process was meticulously planned to minimize disruptions to critical business operations and ensure a smooth transition to the cloud-native infrastructure.

In parallel, comprehensive strategies were devised to handle schema migration, indices replication, and the restoration of auto-incremental sequences post-migration. These strategies were essential to maintain data consistency and integrity throughout the migration process. Rigorous testing and validation procedures were conducted in staging environments to identify and resolve any discrepancies or performance variations before finalizing the migration. The successful migration

to Amazon Aurora empowered the IoT company with a scalable, fault-tolerant database solution, laying the groundwork for future growth and innovation.

The achievements of this migration project were significant. By seamlessly migrating 15 GB of data to Amazon Aurora with zero downtime, the IoT company ensured uninterrupted service delivery to its users. The implementation of AWS Database Migration Service (DMS) facilitated efficient replication of ongoing changes to the target endpoint, ensuring data consistency and integrity. Moreover, the meticulous configuration of source database servers and the adoption of best practices in schema migration and indices replication minimized post-migration disruptions and optimized system performance. Overall, the successful migration to Amazon Aurora enhanced the scalability, fault tolerance, and management efficiency of the IoT company's database infrastructure, setting the stage for future innovation and growth in the cloud-native ecosystem.

Responsibilities:

- Led the strategic initiative to migrate the PostgreSQL standalone instance to Amazon Aurora global
- Conducted thorough configuration of the source PostgreSQL database server and Aurora cluster in AWS RDS.
- Orchestrated the setup and configuration of AWS DMS for seamless data migration.
- Designed and implemented strategies to handle schema migration, indices replication, and fixing auto-incremental sequences.
- Collaborated with cross-functional teams to ensure the compatibility of applications with the new Aurora-managed infrastructure.
- Maintained comprehensive documentation and version control throughout the migration process.
- Ensured adherence to security best practices, including AWS VPC security group configurations.
- Oversaw the validation and finalization of the migration, including the decommissioning of the old PostgreSQL standalone instance.

Configuring Redis Cache for Optimizing Cost and Boosting Performance of RDS for MySQL

Role DevOps Engineer
Technologies AWS (ElastiCache, Redis, Redis Client API, RDS MySql, CloudWatch)

Description:

Configured Redis as a caching solution to optimize costs and enhance the performance of RDS for MySQL. Leveraging Amazon ElastiCache for Redis, we aimed to address the challenge of balancing database costs while improving application performance and response times, especially as data volumes and user bases expanded.

This project commenced with a thorough analysis and planning phase. We scrutinized the data access patterns and query frequencies of the application to identify opportunities for caching. By understanding the application's requirements and workload characteristics, we devised a comprehensive caching strategy tailored to its specific needs. This involved determining which data could benefit from caching, defining cache eviction policies, and estimating potential cost savings and performance improvements.

The implementation phase involved the seamless integration of Amazon ElastiCache for Redis with the existing RDS for MySQL infrastructure. We configured Redis clusters to efficiently cache query results, offloading database IOPS and reducing latency. Custom cache invalidation strategies were devised to ensure data consistency between the Redis cache and the underlying MySQL database. Additionally, we implemented monitoring and alerting mechanisms to track cache health, performance metrics, and resource utilization in real-time, enabling proactive maintenance and optimization.

Significant achievements were realized in terms of cost optimization and performance enhancement. The project resulted in substantial cost savings, with potential reductions of up to 55% compared to using RDS for MySQL alone, translating to approximately $960/month savings. Moreover, read performance saw a remarkable boost, with potential gains of up to 80 times faster read performance. By leveraging Redis caching, the application achieved microsecond response times, supporting massive volumes of throughput efficiently. Additionally, the implementation of caching strategies such as lazy loading and write-through caching further improved application responsiveness and scalability, ensuring instantaneous user response times even under peak loads.

By strategically configuring Redis caching, we enabled our client to better serve their customers, meet growing demands, and expand their market footprint in a cost-effective manner.

Responsibilities:

- Conducted a comprehensive analysis of the application's data access patterns and query frequencies to identify caching opportunities.
- Defined caching strategies based on workload characteristics, determining which data would benefit from caching and establishing cache eviction policies.
- Analyzing the potential cost savings and performance improvements to set project objectives.
- Configured Amazon ElastiCache for Redis clusters to seamlessly integrate with the existing RDS for MySQL infrastructure.
- Ensured compatibility with open-source Redis and leveraged ElastiCache's managed service features for ease of maintenance.
- Developed and implemented lazy loading caching strategy to populate the cache only when requested by the application, reducing unnecessary database trips.
- Deployed write-through caching mechanism to maintain data consistency between the Redis cache and the MySQL database, ensuring accurate and up-to-date information.
- Utilized Redis client API to invoke ElastiCache caching layer, facilitating efficient data retrieval and storage.
- Implemented monitoring and alerting mechanisms to track cache health, performance metrics, and resource utilization in real-time.
- Conducted periodic reviews and performance tuning to ensure continued efficiency and effectiveness of the caching solution.

Chat Message Migration and Real-time Processing Automation

Role DevOps Engineer

Technologies AWS (ECS, SQS, Lambda, DocumentDB, ElastiCache for Redis) Terraform, Python, AWS SDK (Boto3)

Description:

The project aimed to address the challenge of ensuring the durability and persistence of chat messages in a real-time chat server environment while maintaining performance. Initially, messages were stored in Redis for quick access but were temporary and risked being lost if the server restarted. The goal was to implement a solution to move messages from Redis to a suitable database for long-term storage without compromising chat performance.

Developed a comprehensive automation solution leveraging various AWS services and technologies. Recognizing the critical need for permanent message storage, I spearheaded the initiative to transition from Redis to DocumentDB, ensuring message durability and availability beyond the lifespan of the Redis cache. Terraform played a pivotal role in this transition, as configuration files were created for infrastructure as code, streamlining deployment and management processes.

DocumentDB was chosen as the optimal solution for storing chat messages since its seamless compatibility with MongoDB, coupled with its flexible document model capable of handling JSON-like structures, perfectly suited for ensuring message persistence while maintaining high performance and availability.
A Lambda function was meticulously designed and implemented to retrieve messages from SQS and securely store them in DocumentDB. This solution not only addressed the immediate challenge of message persistence but also provided a scalable architecture capable of seamlessly handling future message volumes.
SQS played a pivotal role in managing the flow of messages, with the Lambda function triggered promptly upon message arrival. This orchestration ensured minimal impact on chat server performance, maintaining the real-time nature of the chat application.

The implementation involved crafting the Lambda function to efficiently retrieve and process messages from SQS, applying sophisticated filtering based on channel ID, and seamlessly storing them in DocumentDB. Leveraging Python and the AWS SDK (Boto3), I ensured compatibility and seamless integration with AWS services, laying the foundation for a robust and scalable solution.

Responsibilities:

- Led the initiative to transition chat message storage from SQS to DocumentDB, overseeing the project lifecycle from planning to execution.
- Identified DocumentDB as the optimal solution for storing chat messages, citing its compatibility with MongoDB and flexible document model.
- Collaborated closely with developers to enhance the integration between SQS and the backend, improving communication efficiency and message handling.
- Designed the Lambda function to efficiently retrieve and process messages from SQS, applying sophisticated filtering based on channel ID before storing them in DocumentDB.
- Leveraged Python and the AWS SDK (Boto3) for seamless integration with AWS services, ensuring compatibility and scalability.
- Utilized Terraform to create configuration files for infrastructure as code, streamlining deployment and management processes, including deploying and configuring DocumentDB based on project requirements.
- Conducted thorough testing and validation of the automation solution to guarantee reliability and efficiency.

- Collaborated with the team to integrate the solution with existing ECS-based chat server infrastructure, ensuring compatibility and scalability without disrupting ongoing operations.
- Documented the functionality, usage instructions, and troubleshooting guidelines for the automation solution, facilitating easy adoption and knowledge transfer within the team.

Custom Integration: Sending EKS Events to Splunk for Long-Term Storage and Debugging

Role SRE / DevOps Engineer
Technologies AWS (EKS), Kubernetes, Helm, Crd, Eventtailer Fluentd, Splunk.

Description:

Configured advanced monitoring for EKS pods by integrating them with Splunk for enhanced visibility and long-term debugging capabilities. The goal was to collect and forward pod events from the EKS cluster to Splunk for analysis and storage, addressing the need for comprehensive monitoring beyond default Kubernetes capabilities.

To achieve this, the project involved setting up Fluentd in Kubernetes for log collection, aggregation, and forwarding using Helm charts. This streamlined approach ensured efficient handling of log data within our Kubernetes environment, laying the foundation for enhanced monitoring capabilities.

Additionally configured a Custom Resource Definition (CRD) to facilitate the provisioning of EventTailers, responsible for listening to Kubernetes events and transmitting their changes to stdout. This setup enables the Logging operator to process the events effectively. Additionally, integrated EventTailers seamlessly into our Kubernetes environment, enabling efficient event transmission to Splunk. This comprehensive configuration enhances monitoring and troubleshooting capabilities, ensuring timely insights into the operational dynamics of our infrastructure.

Responsibilities:

- Designed and implemented a custom integration to send EKS pod events to Splunk for long-term storage and debugging.
- Collaborated with developers to identify specific pod events required for monitoring and troubleshooting.
- Configured Fluentd in Kubernetes for log collection, aggregation, and forwarding using Helm charts.
- Created a Kubernetes CRD for provisioning resources necessary for event transmission, extending Kubernetes capabilities to accommodate specialized use cases.
- Deployed EventTailers to enable the collection and forwarding of pod events to Splunk for analysis and storage.
- Conducted thorough testing and validation of the integration to ensure accuracy and reliability of the collected pod events in Splunk.
- Documented the integration process, including step-by-step instructions, configuration details, and troubleshooting guidelines, to facilitate knowledge sharing and maintainability.

Kubernetes with Ansible: Automating Application Deployment on Kubernetes with Ansible

Role

Technologies

DevOps Engineer
AWS (EKS, ECR), Ansible(roles,playbooks,group_vars)
GitHub Actions, Docker, Helm , Jinja.

Description:

Implemented an automated deployment workflow using Ansible, Docker, Kubernetes, and GitHub Actions for CI/CD. Orchestrated seamless deployment pipelines, configured Ansible roles for Docker image building and AWS ECR pushing, and optimized deployment strategies.

The project commenced with designing and implementing a highly efficient deployment workflow. Leveraging Ansible, Docker, Kubernetes, and GitHub Actions, we created a seamless CI/CD pipeline to facilitate swift and reliable application deployment. Central to our approach was the use of Jinja templating within Ansible roles and playbooks, ensuring adaptability across various environments. Ansible served as the backbone, enabling configuration for different deployment tasks with flexibility. Docker played a crucial role in containerizing applications, with tailored Ansible roles for image building and AWS ECR pushing, designed for flexibility across AWS environments.

Within the Kubernetes ecosystem, deployment playbooks embraced Jinja templating for configuration flexibility. These playbooks were seamlessly integrated into GitHub Action pipelines, ensuring consistent and reliable deployment across environments. Continuous optimization of deployment strategies was achieved through performance monitoring and analysis, enabling iterative enhancements to automation workflows, thereby reducing downtime and improving efficiency.

Responsibilities:

- Designing and implementing the automated deployment workflow, ensuring flexibility and scalability.
- Configuring Ansible roles for Docker image building and AWS ECR pushing, making them adaptable to various environments.
- Developing Kubernetes deployment and service playbooks tailored for the project's requirements.
- Utilizing Jinja templating within deployment playbooks to ensure adaptability across diverse environments.
- Integrating all playbooks and roles into GitHub Action pipelines for automated deployment processes.
- Monitoring and optimizing deployment strategies and automation workflows to improve efficiency and reliability.
- Collaborating with team members to address deployment challenges and enhance automation processes, ensuring adaptability to evolving project requirements.

Migrating Application from ECS to Kubernetes

Role

DevOps Engineer

Technologies

AWS (EKS, ECS, ECR), Terraform, Helm, Kubernetes, Nginx Ingress Controller, GitHub Actions

Description:

Led the successful migration project from AWS ECS to Kubernetes (AWS EKS) while implementing a robust CI/CD pipeline and Infrastructure as Code (IaC) practices. The project encompassed the seamless integration of multiple technologies to ensure efficient and reliable application deployment and management.

The project started with designing and planning, addressing the challenges encountered in the existing AWS ECS setup. This involved evaluating various platform and tooling choices to ensure the successful migration to Kubernetes (AWS

EKS). Key decisions included opting for AWS Managed Kubernetes (EKS) over self-managed clusters, selecting Terraform for cluster setup, and determining the appropriate ingress controller.

The deployment of applications posed significant challenges due to the intricacies of transitioning from ECS to Kubernetes. Challenges included adjusting deployment methodologies to accommodate Kubernetes' architecture and resolving compatibility issues with existing applications. To mitigate these challenges, we leveraged Helm charts for streamlined application packaging and deployment. Additionally, we collaborated closely with application teams to refactor and migrate services to Kubernetes-compatible formats, ensuring compatibility and reliability throughout the process.

To minimize downtime, we carefully managed the migration process, reducing DNS TTL for faster updates and employing an automated playbook for switchover and rollback procedures. This allowed us to complete the migration with only ~3 minutes of disruption. Additionally, we addressed ingress controller challenges by transitioning to nginx-ingress-controller, ensuring seamless traffic routing to Kubernetes pods and enhancing application reliability.

Responsibilities:

- Leading discussions on platform choices (e.g., AWS EKS vs self-managed clusters), tooling decisions, and architectural considerations.
- Analyzing the potential cost savings and resource optimizations achievable through the migration to Kubernetes.
- Identifying potential risks associated with the migration, such as downtime, service disruptions, and compatibility issues.
- Developed and managed the CI/CD pipeline using GitHub Actions, enabling automated builds and deployments of container images to AWS ECR.
- Configuring Kubernetes clusters using Terraform, ensuring alignment with best practices and security standards.
- Collaborating with application teams to refactor and migrate services from ECS to Kubernetes-compatible formats (e.g., Helm charts).
- Investigated and resolved challenges related to ingress routing, transitioning from aws-alb-ingress-controller to nginx-ingress-controller to address intermittent 503 errors post-migration.
- Orchestrated the migration process with minimal downtime, employing DNS TTL adjustments and phased deployment strategies to mitigate user impact.
- Automated switchover and rollback procedures, leveraging scripts and playbooks to ensure seamless transitions between ECS and EKS environments.
- Monitored post-migration performance and addressed issues such as CPU throttling through iterative tuning of resource allocations and container configurations.
- Conducting thorough testing in staging environments to ensure compatibility, performance, and reliability of migrated services.
- Documented migration procedures, best practices, and lessons learned for knowledge sharing and future reference to educate team members on Kubernetes concepts, best practices, and operational procedures

Cost Optimization: Efficiency Through Automation

Role FinOps / DevOps Engineer

Technologies AWS (EKS, RDS, OpenSearch, ElastiCache, Amazon MQ, CloudWatch, CloudFormation), Python

Description:

Led a comprehensive cost optimization initiative within the E-commerce sector, with a primary focus on non-production environments. Conducted an in-depth analysis that uncovered instances of overprovisioned resources, including RDS, ElasticSearch, ElastiCache, Amazon MQ, and EKS clusters. Employed precise calculations to strategically resize these resources to smaller instance types, ensuring consistent performance while achieving substantial cost reductions.

Implemented a sophisticated Python-based automation system deployed on AWS Lambdas to manage RDS instances intelligently. This automation efficiently halts and restarts RDS instances during weekends and after working hours, optimizing resource utilization during periods of inactivity. Addressed efficiencies in ElasticSearch and ElastiCache clusters through specialized automation, dynamically adjusting instance types to more cost-effective sizes during non-operational periods.

Achieved a noteworthy 61.3% reduction in RDS expenses and a commendable 20.1% savings for ElastiCache and ElasticSearch clusters. Developed an automation solution to scale down EKS node groups to a minimum during weekends, optimizing cloud infrastructure and minimizing operational costs.

Responsibilities:

- Led the entire project lifecycle, overseeing planning and execution for successful outcomes.
- Identified instances of overprovisioning, conducting detailed cost analyses for critical resources.
- Implemented strategic changes, resizing instance types for RDS, ElasticSearch, ElastiCache, Rabbit MQ, and EKS clusters.
- Introduced a Python-based automation system on AWS Lambdas for efficient RDS instance management.
- Developed innovative automation optimizing resource utilization during weekends and after working hours.
- Addressed ElasticSearch and ElastiCache efficiencies through specialized automation.
- Monitored system performance to ensure cost optimization did not compromise overall efficiency.
- Evaluated results to ensure significant cost reduction without compromising system performance.
- Achieved a remarkable 61.3% reduction in RDS expenses and a commendable 20.1% savings for ElastiCache and ElasticSearch cluster
- Implemented automation to scale down EKS node groups during weekends, optimizing cloud infrastructure and minimizing operational costs.
- Demonstrated a consistent track record of implementing cost-effective solutions and leveraging automation for enhanced cloud efficiency

Kubernetes Infrastructure Enhancement using best practices

Role DevOps Engineer
Technologies AWS (EKS, IAM,), Kubernetes (PDB, Network policies, Node/Pod affinities, HPA

Description:

As the lead DevOps Engineer, I spearheaded a comprehensive project aimed at planning and implementing advanced features to enhance the security, flexibility, and availability of our Kubernetes infrastructure. This initiative involved a meticulous evaluation of our existing setup, identification of potential vulnerabilities, and the deployment of robust security measures to fortify our defenses.

Our primary focus was on implementing various security features within Kubernetes. We began by defining Network Policies to control traffic flow between pods and external resources, thereby reducing the attack surface and enhancing

security. Additionally, Pod Security Policies (PSPs) were enforced to prevent potential security risks arising from overly permissive pod configurations, ensuring a secure runtime environment.

To ensure fault tolerance and minimize downtime, we leveraged Kubernetes' Pod Disruption Budgets (PDB). This resource effectively limits the number of pods of a replicated application that can be down simultaneously during maintenance or infrastructure upgrades, maintaining overall service availability and reliability.

For flexible scheduling and optimal resource allocation, we utilized Node Affinity, Pod Affinity, and Pod Anti-Affinity. These features allowed us to specify rules for pod placement based on custom labels, ensuring efficient resource utilization and scheduling within the Kubernetes cluster.

Furthermore, we implemented Karpenter and Hpa for autoscaling, enabling dynamic scaling of resources without relying on traditional ASG abstraction. Karpenter communicates directly with the cloud API to request EC2 instances with the required resources, optimizing resource allocation and enhancing scalability and Autoscaling capabilities were enhanced using Karpenter for cluster autoscaling and HorizontalPodAutoscaler for dynamic adjustment of resource allocation based on workload metrics. Throughout the project, adherence to security best practices and compliance standards was paramount. We conducted regular audits and vulnerability scans to identify and address any remaining security gaps, ensuring the integrity and reliability of our Kubernetes infrastructure.

Responsibilities:

- Conducted a comprehensive evaluation of the existing Kubernetes infrastructure, identifying security vulnerabilities and potential improvements.
- Collaborated with stakeholders to gather requirements and align security measures with business goals
- Developed a detailed implementation plan outlining necessary actions to enhance security, flexibility, and availability within the Kubernetes environment.
- Implemented Karpenter and HPA for autoscaling, enabling dynamic resource allocation and enhancing scalability.
- Implemented security features such as Network Policies and Pod Security Policies to mitigate potential risks and ensure a secure runtime environment.
- Deployed Pod Disruption Budgets (PDB) to maintain service availability and reliability during maintenance and upgrades.
- Utilized Node Affinity, Pod Affinity, and Pod Anti-Affinity for flexible scheduling and optimal resource allocation.
- Developed custom Resource Definitions (CRDs) to extend Kubernetes functionality and accommodate specialized use cases.
- Conducted regular audits and vulnerability scans to identify and address any remaining security gaps, ensuring the integrity and reliability of the Kubernetes infrastructure.
- Actively participated in knowledge-sharing sessions, providing guidance on Kubernetes best practices, automation techniques, and resource optimization to junior team members.

Enhanced Security Implementation Project for GitLab CI Workflow

Role DevSecOps Engineer
Technologies GitLab CI/CD, SAST, DAST, IaC scanning, Secret Detection, Dependency Scanning, Container Scanning

Description:
Led a comprehensive initiative focused on fortifying the security of our GitLab CI workflow. This project involved configuring security best practices, implementing a suite of security tools, and conducting meticulous evaluation, improvement planning, and implementation. We identified and rectified potential vulnerabilities and misconfigurations to enhance our development processes and elevate our overall security posture.

We began by systematically scanning our source code for known vulnerabilities using Static Application Security Testing (SAST). This allowed us to identify and address potential security flaws early in the development cycle. Leveraging SAST analyzers integrated into GitLab CI/CD, we generated JSON-formatted reports as artifacts, facilitating seamless integration into our workflow.

Furthermore, we automated Dynamic Application Security Testing (DAST) to simulate real-world attacks on our web applications and APIs. By automating penetration tests, we uncovered vulnerabilities such as cross-site scripting (XSS), SQL injection (SQLi), and cross-site request forgery (CSRF), enabling us to address them proactively.

In addition to code scanning, we implemented Infrastructure as Code (IaC) scanning for our Terraform and Ansible files. This allowed us to identify vulnerabilities in our infrastructure definitions before they were committed to the default branch, minimizing potential risks.

To prevent sensitive information from being exposed in our repositories, we employed Secret Detection scanning. This helped us identify and revoke exposed secrets like keys or API tokens, reducing the risk of unauthorized access.
For our Docker-based applications, we integrated Container Scanning into our CI/CD pipeline. This enabled us to audit our Docker images for known vulnerabilities and display the results directly in merge requests, ensuring that only secure images were deployed.

Moreover, we utilized Dependency Scanning to analyze our application's dependencies for known vulnerabilities. By scanning all dependencies, including transitive dependencies, we proactively addressed risks associated with third-party libraries and frameworks.

Responsibilities:

- Performed a comprehensive assessment of the GitLab CI workflow, identifying security vulnerabilities and areas for enhancement.
- Developed a detailed implementation plan outlining necessary actions to bolster security across the CI/CD pipeline.

- Implemented a range of security measures including SAST, DAST, IaC scanning, Secret Detection, Dependency Scanning, and Container Scanning.
- Facilitated knowledge-sharing sessions to provide guidance on GitLab CI best practices and security tool utilization to team members.
- Performed regular vulnerability scans and audits to detect and rectify any remaining security gaps in the CI/CD pipeline.
- Documented security configurations, policies, and procedures to create comprehensive reference material for present and future audits.

Seamless Migration and Management of manually and AWS CDK created Infrastructure with Terraform Modules

Role DevOps Engineer

Technologies AWS Cloud Development Kit, Terraform, GitLab CI, AWS Services.

Description:

Led the migration from AWS CDK to Terraform, overseeing a seamless transition that incorporated both CDK-generated and manually created infrastructure. This involved meticulous analysis and documentation of the existing infrastructure landscape, crafting a comprehensive migration plan, and developing Terraform modules to replicate and enhance functionality.

The endeavor commenced with defining a comprehensive strategy to integrate both CDK-generated and manually created infrastructure into Terraform. This involved meticulously documenting dependencies, configurations, and deployment processes, laying the foundation for a smooth migration process.
The migration plan was meticulously crafted to ensure minimal disruption to critical business operations. Leveraging Terraform's declarative language and modular design principles, Terraform modules were developed to replicate and enhance the functionality of existing infrastructure components. These modules were designed to be easily customizable and adaptable, promoting consistency and reducing configuration drift. Rigorous testing and validation were conducted in staging environments to identify and resolve any discrepancies or performance variations. Collaborating closely with cross-functional teams, compatibility with applications was ensured, facilitating a seamless transition to the new Terraform-managed infrastructure.

The project also incorporated Terraform workspaces for organizing environments, facilitating management of configuration variations while maintaining consistency and separation between development, staging, and production environments. Terraform Cloud was leveraged for state management, enabling seamless collaboration and synchronization across the team.Throughout the project, version-controlled documentation was meticulously maintained, ensuring traceability and accountability. Post-migration environments were diligently monitored for stability, with insights and best practices shared to enhance overall infrastructure management.

Responsibilities:

- Led a strategic initiative to transition from AWS CDK to Terraform for orchestrating and managing cloud infrastructure, incorporating both manually created and CDK-based resources.
- Conducted a comprehensive analysis of existing infrastructure, documenting dependencies, resource configurations, and deployment processes.

- Designed and executed a migration plan, ensuring a seamless transition while mitigating potential disruptions to critical business operations, encompassing both manually created and CDK-based resources.
- Developed Terraform modules and configurations to replicate and enhance the functionality of existing infrastructure components, adhering to industry best practices and leveraging Terraform's declarative language.
- Conducted rigorous testing and validation of the Terraform-based infrastructure in staging environments, identifying and resolving any discrepancies or performance variations.
- Collaborated closely with cross-functional teams, providing guidance and support during the migration and ensuring compatibility of applications with the new Terraform-managed infrastructure.
- Incorporated Terraform workspaces for organizing environments, facilitating management of configuration variations while maintaining consistency and separation between environments.
- Leveraged Terraform Cloud for storing state, enabling seamless collaboration and ensuring synchronization across the team.
- Integrated the Terraform modules into CI/CD pipelines and deployment workflows, automating the infrastructure provisioning process.
- Shared insights and best practices with the team, facilitating knowledge transfer and fostering a culture of continuous improvement in infrastructure management.

Implementing GitOps with ArgoCD: Automating Application Deployment on Kubernetes

Role DevOps Engineer

Technologies AWS (EKS), GCP (GKE), GitOps, ArgoCD, Kubernetes, Helm, Kustomize

Description:

Implemented GitOps practices using ArgoCD, revolutionizing the way applications were deployed and managed on Kubernetes. This project involved leveraging Git as the single source of truth for declaratively managing the desired state of applications and infrastructure, enabling automated, reliable, and auditable deployment workflows.

The project began by designing and implementing an automated process for creating ArgoCD applications for existing Helm charts and Git repositories. This involved defining the necessary configurations and templates to ensure seamless integration between ArgoCD and the application repositories. By leveraging Helm, the popular package manager for Kubernetes, we achieved consistency and repeatability in application deployments.

Central to this project was the setup of a multi-environment ArgoCD setup, enabling the management of application deployments across different environments. With the use of Kustomize, a native Kubernetes configuration management tool, we were able to manage environment-specific configurations and ensure proper separation of concerns. This allowed for the streamlined deployment of applications across various environments, such as development, staging, and production.

Responsibilities:

- Designed and implemented the GitOps strategy using ArgoCD, enabling declarative management of application deployments on Kubernetes.

- Developed automated processes for creating ArgoCD applications, seamlessly integrating with existing Helm charts and Git repositories.
- Define GitOps workflows, establishing Git as the single source of truth for application deployments.
- Configured and maintained the ArgoCD setup, ensuring high availability, security, and scalability of the platform.
- Implemented multi-environment ArgoCD setups, managing deployments across different environments using Kustomize for environment-specific configurations.
- Establish best practices for managing infrastructure as code within the GitOps framework.
- Conducted extensive testing and validation of the GitOps workflows, ensuring reliability, consistency, and fault tolerance.
- Developed and maintained documentation for the GitOps processes, including configuration guidelines, troubleshooting steps, and best practices.
- Provided guidance and support to the development and operations teams, facilitating the adoption of GitOps practices and ArgoCD.
- Conducted training sessions to educate team members on GitOps principles, ArgoCD usage, and best practices for application deployment on Kubernetes.