Master Course on Cloud-Native DevOps with Harness

Module 1: Introduction to Harness and Cloud-Native DevOps

Topics:

- Overview of Cloud-Native DevOps practices.
- Introduction to Harness: Key features and benefits.
- Setting up Harness for the first time.Project: Set up ArgoCD in a Minikube cluster.

Module 2: Multi-Cloud Deployments with Harness

Topics:

- Understanding multi-cloud strategies.
- Configuring Harness for AWS, GCP, and Azure deployments.

Module 3: Advanced Deployment Strategies

Topics:

- Principles of blue/green and canary deployments.
- Implementing zero-downtime deployment strategies in Harness.

Module 4: Automated Testing and Quality Gates

Topics:

- Integrating testing frameworks with CI/CD pipelines.
- Setting up pre-deployment and post-deployment quality gates in Harness.

Module 5: Security in Continuous Deployment

Topics:

- Foundations of Secure Deployments: Understanding the security challenges in continuous deployment environments and the principles of secure by design.
- Secrets Management with HashiCorp Vault: Deep dive into using HashiCorp Vault for managing secrets, keys, and tokens securely. Overview of Vault's architecture, authentication methods, and secret engines.
- Integrating Harness with HashiCorp Vault: Step-by-step guidance on configuring Harness to securely fetch secrets from HashiCorp Vault, enabling secure application configurations without exposing sensitive information in the deployment process.

- Encryption and Compliance: Implementing encryption in transit and at rest, understanding compliance requirements relevant to continuous deployment, and how to meet them.
- Vulnerability Scanning and Remediation: Incorporating automated vulnerability scanning into CI/CD pipelines, interpreting scan results, and implementing remediation strategies.
- Monitoring and Auditing for Security: Setting up monitoring and auditing mechanisms to detect security incidents early and conducting post-incident analyses to prevent future breaches.

Module 6: Microservices Deployment Optimization

Topics:

- Introduction to Microservices Deployment Challenges: Overview of common challenges in deploying and managing microservices, including service discovery, configuration management, and inter-service communication.
- Harness for Microservices Deployment: Utilizing Harness to automate and manage deployments across microservices architectures, focusing on CI/CD pipelines, environment variables, and deployment strategies tailored for microservices.
- Canary Releases in Microservices: Implementing canary releases with Harness to minimize the impact of new deployments on the overall system, allowing for gradual rollout and testing in production-like environments.
- Service Dependency Management: Strategies for managing dependencies between microservices during deployments, including versioning, backward compatibility, and database schema migrations.
- Optimizing Microservices Deployments with Harness: Best practices for using Harness to achieve efficient, reliable, and scalable deployments of microservices, leveraging features such as blue/green deployments, automated rollbacks, and monitoring integrations.

Module 7: Monitoring and Observability

Topics:

- Foundations of Monitoring and Observability: Differentiating between monitoring and observability, and understanding their importance in modern DevOps practices.
- Configuring Monitoring and Logging with Harness: Practical guidance on setting up comprehensive monitoring and logging for applications deployed via Harness, using tools like Prometheus, Grafana, and ELK Stack.
- Implementing Observability in Deployments: Techniques for achieving deep observability into application health, performance, and user experiences through distributed tracing, metrics collection, and log aggregation.
- Using Observability Data to Inform Deployment Strategies: How to use real-time data and insights gathered from observability tools to make informed decisions about deployment frequencies, scaling, and rollback mechanisms.

- Best Practices for Observability: Strategies for maintaining an effective observability stack, including log management policies, metric selection, and visualization techniques.

Module 8: Scaling and Performance Tuning

Topics:

- Principles of Scalability in Cloud-Native Applications: Understanding the fundamentals of scaling applications and infrastructure in a cloud-native ecosystem, including horizontal vs. vertical scaling and the role of microservices and serverless architectures.
- Automating Scaling with Harness: How to leverage Harness to automate the scaling of applications and infrastructure, including the use of auto-scaling groups, containers orchestration platforms like Kubernetes, and serverless functions.
- Performance Tuning Techniques: Methods for tuning the performance of cloud-native applications, focusing on optimizing code, leveraging caching, managing resource allocations, and minimizing latency.
- Monitoring and Metrics for Scaling Decisions: Utilizing monitoring tools and metrics to make informed scaling decisions, including the identification of bottlenecks and performance thresholds.
- Best Practices for Scaling and Performance Tuning: Strategies to ensure effective scaling and performance tuning, including continuous testing, load balancing, and the importance of a proactive approach to performance issues.

Deliverables:

- Comprehensive lecture notes and slides for each module.
- Step-by-step guides for hands-on projects.
- Access to a forum for Q&A and discussions.
- Final assessment test to evaluate understanding and practical skills.

Certificate of completion.

This course structure is designed to provide a thorough understanding of Harness, from basic concepts to advanced practices, with a strong focus on real-world applications and projects that prepare students for implementing Harness in their Production environments effectively.

Hands-on Projects

Project 1: Automating Multi-Cloud Deployments with Harness

Problem Statement:

In today's rapidly evolving digital landscape, managing deployments across multiple cloud environments can be challenging due to varying APIs, services, and configurations. As a Solution Architect at a leading software company, you're tasked with leveraging Harness to automate and streamline the deployment process for a highly available, scalable web application across AWS, GCP, and Azure. Your objective is to create a seamless workflow that supports consistent deployments, enables quick rollbacks in case of issues, and maintains high reliability and uptime across different environments.

Objective:

Design and implement an automated deployment pipeline using Harness that deploys a multi-container application across AWS, GCP, and Azure, ensuring high availability, scalability, and consistent configuration across cloud providers.

Task Breakdown

Task 1: Defining the Deployment Architecture

Objective: Outline the multi-cloud deployment architecture and identify the components of the web application to be containerized and deployed.

Activities:

- Analyze the application architecture to determine the microservices to be deployed and their dependencies.
- Design a deployment strategy that includes blue/green or canary deployments for minimizing downtime and risk.

Task 2: Containerizing the Application

Objective: Prepare the application for deployment by creating Docker containers for each microservice.

Activities:

- Write Dockerfiles for each component of the application, ensuring best practices for security and efficiency.
- Build and push the Docker images to a container registry accessible by AWS, GCP, and Azure.

Task 3: Configuring Harness for Multi-Cloud Deployments

Objective: Set up Harness Continuous Delivery to manage deployments across multiple cloud platforms.

Activities:

- Integrate Harness with AWS, GCP, and Azure, setting up the necessary credentials and permissions for accessing resources.
- Define the Harness deployment pipelines, specifying the environments, infrastructure definitions, and deployment strategies.

Task 4: Implementing Advanced Deployment Strategies

Objective: Utilize Harness to implement advanced deployment strategies across cloud environments.

Activities:

- Configure blue/green deployments in Harness to enable zero-downtime updates and instant rollbacks.
- Set up canary deployment workflows to gradually roll out changes to a small subset of users before a full-scale launch.

Task 5: Monitoring and Optimization

Objective: Monitor the deployments and optimize the process for efficiency and reliability.

Activities:

- Integrate Harness with monitoring tools like Prometheus or Datadog to track the performance of deployments.
- Analyze deployment data to identify bottlenecks or issues and refine the deployment strategies for better performance and reliability.

Deliverables:

- A comprehensive multi-cloud deployment workflow implemented in Harness, complete with documentation on the setup, configuration, and deployment strategies.
- Dockerfiles and CI/CD configurations for the application, demonstrating best practices in containerization and automated deployments.
- A report on the deployment performance across different cloud environments, including insights on reliability, uptime, and areas for optimization.

==========

Project 2: Implementing Blue/Green Deployments for Zero-Downtime Updates

Problem Statement:

You are a DevOps Engineer tasked with implementing a blue/green deployment strategy for a critical production service using Harness. The project involves automating the infrastructure provisioning, application deployment, and traffic switching to ensure zero downtime during updates. The ability to quickly revert to the previous version is paramount in case issues arise during the deployment.

Objective:

Design and execute a blue/green deployment strategy using Harness to automate the entire process for a critical production service, ensuring seamless updates and instant rollback capabilities.

Task Breakdown

Task 1: Infrastructure Provisioning

Objective: Automatically provision and configure the necessary infrastructure for both blue and green environments.

Activities:

- Use Harness' Infrastructure as Code capabilities to define infrastructure requirements.
- Automate the provisioning of identical production environments (blue/green) using cloud services.

Task 2: Application Deployment Automation

Objective: Set up Harness pipelines for automated application deployments to blue/green environments.

Activities:

- Configure application services in Harness, specifying deployment artifacts and parameters.
- Create deployment pipelines in Harness, integrating pre-deployment approvals and automated tests.

Task 3: Traffic Management and Switching

Objective: Implement traffic management to switch between blue and green environments seamlessly.

Activities:

- Utilize Harness' traffic management features to control the flow of user traffic to the new (green) environment.
- Define criteria and automated checks that trigger the switch from blue to green.

Task 4: Rollback Mechanism

Objective: Establish an automated rollback process to the previous version in case of deployment failure.

Activities:

- Configure automatic rollback in Harness based on predefined failure criteria or manual intervention.
- Test the rollback process to ensure quick reversion to the blue environment if needed.

Deliverables:

- A comprehensive report detailing the setup and execution of the blue/green deployment strategy, including infrastructure provisioning, application deployment automation, traffic management, and rollback mechanism using Harness.
- Documentation of the Harness configurations, pipeline definitions, and criteria for traffic switching and rollback.
- Analysis of the deployment process, highlighting the benefits of blue/green deployments in maintaining service availability and user experience.

Project 3: Integrating Automated Testing and Quality Gates with Harness

Problem Statement :

In an effort to enhance the software development lifecycle for a FinTech application, you, as a DevOps engineer, are tasked with integrating automated testing and quality gates within the CI/CD pipeline. Using Harness, you must ensure that code deployments to production are of the highest quality, meeting strict regulatory standards and customer expectations.

Objective:

Implement an end-to-end CI/CD pipeline in Harness that integrates automated testing frameworks and establishes quality gates to ensure that only high-quality code is deployed to production.

Task Breakdown

Task 1: Integrating Automated Testing Frameworks

Objective: Seamlessly integrate automated testing frameworks into the CI/CD pipeline.

Activities:

- Configure Harness to trigger automated unit and integration tests using preferred frameworks (e.g., JUnit for Java applications) after every commit.
- Set up environment-specific configurations to run end-to-end tests in a staging environment.

Task 2: Establishing Pre-deployment Quality Gates

Objective: Create pre-deployment quality gates in Harness to evaluate code quality before production deployment.

Activities:

- Define criteria for passing the quality gates, including test coverage thresholds, static code analysis metrics, and security scan results.
- Configure Harness to automatically halt deployments if the code fails to meet the quality gate criteria.

Task 3: Implementing Post-deployment Quality Gates

Objective: Set up post-deployment quality gates to monitor application health and performance in production.

Activities:

- Utilize Harness to monitor key performance indicators (KPIs) and user feedback post-deployment.
- Establish automated rollback mechanisms triggered by post-deployment gate failures, such as performance degradation or increased error rates.

Task 4: Feedback Loops and Continuous Improvement

Objective: Utilize testing and quality gate results to foster continuous improvement in code quality.

Activities:

- Set up notifications and dashboards in Harness to provide developers with immediate feedback on test failures and quality gate outcomes.
- Facilitate regular review sessions to discuss the outcomes and plan improvements.

Deliverables:

- A fully functional CI/CD pipeline in Harness that integrates automated testing and enforces quality gates.
- Documentation detailing the setup process, testing framework integrations, quality gate criteria, and feedback mechanisms.
- An analysis report summarizing the impact of integrated testing and quality gates on code quality and deployment reliability.
- This project within Module 4 equips students with practical experience in enhancing CI/CD pipelines with automated testing and quality gates using Harness. By completing this project, students will learn how to maintain high standards of code quality, ensuring reliable and efficient software delivery.

==========

Project 4: Automated Testing and Quality Gates

Objective:

This module delves into the critical practice of incorporating automated testing within the CI/CD pipeline to ensure that every code change is validated before being deployed. Students will learn how to integrate various testing frameworks with Harness and set up both pre-deployment and post-deployment quality gates to maintain high standards of code quality and application reliability.

Topics Covered:

- Introduction to Automated Testing in CI/CD: Importance of automated testing, types of tests (unit, integration, end-to-end).
- Integrating Testing Frameworks: Techniques for integrating testing frameworks (e.g., JUnit, Selenium, Jest) with CI/CD pipelines in Harness.
- Quality Gates in Harness: Setting up pre-deployment and post-deployment quality gates in Harness to automatically enforce code quality standards.
- Feedback Loops for Continuous Improvement: Utilizing test results and quality gate outcomes to create feedback loops for developers, enhancing code quality over time.

==========

Project 5: Enhancing Deployment Security with Harness and HashiCorp Vault

Overview:

This module addresses the critical importance of security in the continuous deployment process, focusing on best practices for secure deployments and the integration of Harness with HashiCorp Vault for secure secret management. Students will learn how to safeguard deployment pipelines and maintain confidentiality, integrity, and availability of services throughout the CI/CD process.

Problem Statement :

As a Security Engineer at a financial technology company, you are tasked with enhancing the security of the application deployment process. The current CI/CD pipeline lacks robust secret management and exposes sensitive data, posing significant security risks. Your goal is to integrate Harness with HashiCorp Vault to secure the management and usage of secrets across the deployment lifecycle, ensuring that sensitive information is handled securely and in compliance with financial industry regulations.

Objective:

Implement a secure continuous deployment pipeline using Harness integrated with HashiCorp Vault, focusing on secure secret management and adherence to security best practices.

Task Breakdown

Task 1: Setting Up HashiCorp Vault

Objective: Deploy and configure HashiCorp Vault as the central secrets management solution.

Activities:

- Install and initialize HashiCorp Vault, configuring it for high availability and secure access.
- Define policies and roles in Vault to control access to secrets based on the principle of least privilege.

Task 2: Integrating Vault with Harness

Objective: Configure Harness to securely retrieve secrets from HashiCorp Vault for deployment processes.

Activities:

- Establish a secure connection between Harness and Vault using Vault's authentication methods.
- Modify the Harness deployment pipelines to fetch secrets from Vault dynamically at runtime, eliminating the need to store sensitive data in source code or CI/CD configurations.

Task 3: Implementing Encryption and Compliance Measures

Objective: Ensure encryption of sensitive data in transit and at rest, aligning with compliance standards.

Activities:

- Configure TLS for all communications between Harness, Vault, and the deployment environments.

- Implement data encryption in Vault and ensure deployment artifacts are encrypted at rest.

Task 4: Vulnerability Management

Objective: Integrate vulnerability scanning tools into the CI/CD pipeline, ensuring continuous security assessment.

Activities:

- Select and integrate a vulnerability scanning tool with the Harness pipeline to scan code and dependencies for known vulnerabilities.
- Set up notifications and automate the triage process for addressing identified vulnerabilities.

Task 5: Security Monitoring and Auditing

Objective: Establish comprehensive security monitoring and auditing for the deployment process.

Activities:

- Integrate security monitoring tools to track and analyze deployment activities for suspicious behavior.
- Configure Vault's audit log to record secret accesses and authentication events, enabling detailed auditing and compliance reporting.

Deliverables:

- A secure CI/CD pipeline configuration that integrates Harness with HashiCorp Vault for secret management, complete with documentation on setup and operations.
- Guidelines and configurations for encryption, compliance adherence, and vulnerability management within the CI/CD process.
- A monitoring and auditing framework that provides visibility into security events and compliance with regulatory standards.
- This project within Module 5 prepares students to tackle security challenges in continuous deployment, ensuring they can build secure, compliant, and efficient CI/CD pipelines for modern cloud-native applications.

Project 6: Optimizing Microservices Deployment with Harness

Overview

In this module, students will explore the complexities and challenges associated with deploying microservices architectures, focusing on how Harness can streamline and optimize these processes. The course will cover strategic approaches to managing service dependencies, implementing canary releases, and utilizing Harness's features to enhance the deployment lifecycle of microservices.

Problem Statement:

As a DevOps Engineer at a technology company with a complex microservices architecture for its flagship product, you encounter various challenges in deploying updates and new features efficiently. The current deployment process is manual, error-prone, and lacks the capability to manage service dependencies effectively. Your objective is to leverage Harness to optimize the deployment process, implement canary releases for safer rollouts, and manage service dependencies seamlessly.

Objective:

Design and implement an optimized deployment process for a microservices-based application using Harness, incorporating canary releases and effective service dependency management to ensure smooth, reliable updates.

Task Breakdown

Task 1: Configuring Harness for Microservices Deployment

Objective: Set up Harness to manage deployments across a microservices architecture.

Activities:

- Define microservices in Harness as individual applications or services, configuring CI/CD pipelines for each.
- Set up environment configurations in Harness to handle different stages of the deployment process, from development to production.

Task 2: Implementing Canary Releases

Objective: Utilize Harness to implement canary release strategies for microservices.

Activities:

- Configure canary deployment strategies in Harness for critical microservices, specifying criteria for gradual traffic shifting and monitoring.
- Integrate monitoring tools with Harness to track the performance of canary releases, setting thresholds for automatic rollback or progression.

Task 3: Managing Service Dependencies

Objective: Develop a strategy for managing dependencies between microservices during deployments..

Activities:

- Utilize Harness to orchestrate deployment sequences that respect service dependencies, ensuring no service is deployed without its dependencies being in the correct state.
- Implement versioning strategies for APIs and shared libraries to maintain compatibility across services.

Task 4: Deployment Optimization and Monitoring

Objective: Optimize the deployment process and set up comprehensive monitoring.

Activities:

- Leverage Harness features such as automated rollbacks, environment variables, and secret management to optimize deployments.
- Configure Harness to use integrated monitoring solutions to track deployment success and microservices health post-deployment.

Deliverables:

- A streamlined deployment process for a microservices architecture, fully managed by Harness, with detailed documentation on the configuration and setup.
- Implementation of canary releases for selected microservices, including criteria for evaluation and rollback mechanisms.
- A comprehensive strategy for managing service dependencies during deployments, ensuring high availability and compatibility.
- An analysis report on the deployment optimization efforts, highlighting improvements in deployment speed, reliability, and system stability.
- This project equips students with practical skills in deploying and managing microservices architectures using Harness, focusing on deployment optimization, canary releases, and service dependency management to address the unique challenges of microservices.

==========

Project 7: Monitoring and Observability

Overview:

This module dives into the essential practices of monitoring and observability within cloud-native deployments, emphasizing the role these practices play in ensuring application performance and reliability. Through detailed exploration of configuring monitoring and logging with Harness and leveraging observability to refine deployment strategies, students will learn to implement a proactive approach to application management in dynamic environments.

Problem Statement :

As a DevOps engineer at an e-commerce platform experiencing rapid growth, you are faced with the challenge of ensuring optimal performance and reliability of the platform, especially during peak shopping seasons. The current monitoring setup provides limited insight into system performance and user experience, making it difficult to anticipate and react to issues proactively. Your task is to enhance the deployment process with advanced observability tools integrated with Harness, enabling real-time visibility into the application's health and performance across all deployment stages.

Objective:

Implement an observability framework using Harness that provides deep insights into application performance, user experience, and system health, utilizing this data to optimize deployment strategies for enhanced reliability and performance.

Task Breakdown

Task 1: Configuring Monitoring and Logging

Objective: Set up a comprehensive monitoring and logging system for the application deployed through Harness.

Activities:

- Integrate Prometheus and Grafana with Harness for metric collection and visualization.
- Configure centralized logging using the ELK Stack (Elasticsearch, Logstash, Kibana) to aggregate logs from various services and deployments.

Task 2: Implementing Application Observability

Objective: Achieve in-depth observability into the application's operational health.

Activities:

- Implement distributed tracing with tools like Jaeger or Zipkin to trace requests across microservices.
- Set up application performance monitoring (APM) to track and analyze user transactions and interactions.

Task 3: Utilizing Observability Data for Deployment Decisions

Objective: Use observability data to inform and refine deployment strategies.

Activities:

- Develop a dashboard in Grafana that displays key performance indicators (KPIs) relevant to deployment decisions, such as response times, error rates, and throughput.
- Establish alerts based on thresholds that indicate potential issues, integrating them with the deployment process in Harness to trigger automatic scaling, rollbacks, or canary deployments..

Task 4: Best Practices and Continuous Improvement

Objective: Implement best practices for observability and use insights for continuous improvement.

Activities:

- Document observability best practices, including log management policies and metric selection.
- Conduct regular reviews of observability data to identify trends, anticipate potential issues, and plan for capacity adjustments.

Deliverables:

- A detailed implementation plan and documentation for setting up monitoring, logging, and observability with Harness.
- Customized dashboards and alerting configurations that provide real-time insights into application health and performance.
- A report outlining how observability data has informed deployment strategies, including examples of improvements in application reliability and performance.

==========

Project 8: Optimizing a High-Traffic Web Application with Harness

Overview:

This module focuses on the strategies and techniques required to effectively scale and tune the performance of cloud-native applications. Emphasizing the use of Harness for automating scaling actions and performance tuning tasks, students will learn how to ensure applications not only meet current demands but are also prepared for future growth and performance expectations.

Problem Statement :

As the lead DevOps engineer for a high-traffic web application, you're encountering challenges in meeting the performance and scalability demands during peak usage times. The application experiences slow response times and occasional downtime, affecting user satisfaction and revenue. Your goal is to utilize Harness to automate scaling actions and apply performance tuning techniques to enhance the application's scalability and overall performance.

Objective:

Develop and implement a comprehensive strategy for scaling and performance tuning of a high-traffic web application using Harness, ensuring the application can handle peak loads efficiently while maintaining optimal performance.

Task Breakdown

Task 1: Assessing Current Scalability and Performance

Objective: Analyze the current application architecture and performance bottlenecks.

Activities:

- Conduct a thorough assessment of the application's architecture, identifying areas where performance and scalability are limited.
- Utilize performance monitoring tools to pinpoint specific bottlenecks in the application's codebase, infrastructure, and dependencies.

Task 2: Implementing Automated Scaling with Harness

Objective: Automate the application and infrastructure scaling process using Harness.

Activities:

- Configure Harness to manage the deployment of scalable resources, such as Kubernetes pods or serverless functions, based on real-time demand.
- Set up auto-scaling policies within Harness that dynamically adjust resources during peak and off-peak periods to optimize costs and performance.

Task 3: Performance Tuning of the Application

Objective: Apply performance tuning techniques to improve application efficiency.

Activities:

- Optimize application code and database queries to reduce latency and improve response times.
- Implement caching strategies and content delivery networks (CDN) to decrease load times for static and frequently accessed content.

Task 4: Monitoring, Testing, and Refinement

Objective: Establish a continuous improvement cycle for scalability and performance.

Activities:

- Enhance monitoring setups to track the impact of scaling and performance tuning efforts, focusing on key performance indicators (KPIs).
- Conduct stress and load testing to validate the application's improved scalability and performance under simulated peak conditions.
- Iterate on scaling strategies and performance optimizations based on testing outcomes and monitoring insights.

Deliverables:

- A detailed strategy and implementation guide for scaling and performance tuning the web application using Harness, including configurations and policies for automated scaling.
- Documentation of performance optimizations applied to the application and infrastructure, with before-and-after performance metrics.
- Reports from monitoring and testing phases that demonstrate the application's improved ability to handle peak loads and deliver an optimal user experience.

EACH PROJECT IS DESIGNED TO SIMULATE SCENARIOS THAT PROFESSIONALS MIGHT ENCOUNTER IN REAL-WORLD SETTINGS, ALLOWING STUDENTS TO APPLY THEIR KNOWLEDGE OF HARNESS AND GITOPS PRINCIPLES TO SOLVE PRACTICAL PROBLEMS.