Advanced Docker and Kubernetes Course for Intermediate Students

Course Overview:

This course aims to elevate the skills of professionals in Docker and Kubernetes, providing in-depth knowledge and practical experience with advanced containerization techniques, Kubernetes cluster management, application deployment, and security. It combines theoretical instruction with extensive hands-on exercises, real-world scenarios, and best practices to prepare students for advanced roles in cloud-native development and operations.

Module 1: Advanced Docker Techniques

- Dockerfile optimizations for smaller, more secure images
- Multi-stage builds and minimizing image vulnerabilities
- Docker networking deep dive: Bridge, Host, Overlay networks
- Managing Docker volumes and persistent data strategies

Module 2: Kubernetes Architecture Deep Dive

- Understanding Kubernetes internals: etcd, API server, scheduler, controller manager
- Advanced pod scheduling: Node/Pod affinity and anti-affinity, Taints and Tolerations
- Kubernetes security: Pod Security Policies, Network Policies, Role-Based Access Control (RBAC)

Module 3: Stateful Applications in Kubernetes

- Managing stateful sets and understanding their lifecycle and update strategies
- Dynamic volume provisioning with Persistent Volume Claims (PVCs) and Storage Classes
- Backup and disaster recovery strategies for stateful applications

Module 4: Kubernetes Networking

- Implementing Ingress resources for HTTP routing
- Network policies for securing pod communication within and across clusters

Module 5: CI/CD with Kubernetes

- Automating deployment pipelines using Jenkins, GitLab CI, or GitHub Actions with Kubernetes integration
- Canary deployments and blue-green deployments in Kubernetes
- Rollbacks and versioned deployments with Helm charts

Module 6: Monitoring, Logging, and Observability

- Implementing monitoring with Prometheus and Grafana
- Centralized logging with Elasticsearch, Fluentd, and Kibana (EFK stack)
- Application tracing with Jaeger or Zipkin in Kubernetes environments

Module 7: Kubernetes Security Best Practices

- Securing cluster components and communication
- Implementing secrets management with Kubernetes Secrets and HashiCorp Vault
- Vulnerability scanning for container images and Kubernetes manifests

Module 8: Performance Tuning and Optimization

- Resource requests and limits for optimal scheduling and resource utilization
- Horizontal Pod Autoscaler (HPA) and Vertical Pod Autoscaler (VPA) for dynamic scaling
- Analyzing and optimizing application performance in Kubernetes environments

==========

Hands-on Projects

Project 1: Multi-Container Application Deployment

Project Overview:

This project is designed to give students a comprehensive, hands-on experience in deploying a multi-container application using Docker and Kubernetes, focusing on utilizing advanced containerization techniques, Kubernetes networking, and persistent storage solutions. The goal is to simulate a real-world scenario where students manage the lifecycle of a complex application, ensuring its resilience, scalability, and security.

Objective:

Deploy a scalable multi-container application on Kubernetes, implementing advanced Docker techniques for image optimization and Kubernetes resources for effective networking and data persistence.

Task 1: Application and Docker Configuration

Objective: Prepare a multi-component application for deployment, optimizing Docker configurations for production.

Activities:

- Containerize Application Components: Write Dockerfiles for each component of the application, such as a front-end web interface, a back-end API, and a worker process. Employ multi-stage builds to reduce image sizes.
- Optimize Images: Apply best practices to minimize the size and increase the security of the Docker images. Use tools like Docker's buildx for building multi-platform images and scanners like Trivy for identifying vulnerabilities.
- Container Registry: Push the optimized Docker images to a container registry like Docker Hub, Google Container Registry (GCR), or Amazon Elastic Container Registry (ECR).

Task 2: Kubernetes Deployment and Networking

Objective: Deploy the application on Kubernetes, configuring networking to enable component interaction and external access.

Activities:

- Kubernetes Manifests: Create Kubernetes manifests for each application component, defining Deployments, Services, and Ingress resources as needed. Use ConfigMaps and Secrets to manage configuration and sensitive data.
- Ingress Controller: Set up an Ingress controller to manage external access to the application. Configure Ingress rules to route traffic to the appropriate services based on the request path or host.
- Service Discovery: Implement internal service discovery mechanisms to enable communication between application components. Utilize Kubernetes DNS for service resolution.

Task 3: Implementing Persistent Storage

Objective: Ensure data persistence for stateful components of the application using Kubernetes Persistent Volumes (PVs) and Persistent Volume Claims (PVCs).

Activities:

- Persistent Volume Claims: Create PVCs for stateful application components that require persistent storage, such as databases or file storage.
- Storage Classes: Utilize dynamic volume provisioning by defining Storage Classes tailored to different data storage needs (e.g., SSD-based storage for high performance).
- StatefulSets: For database components, consider using StatefulSets to manage the deployment and scaling of a set of Pods, ensuring orderly deployment, scaling, and termination.

Task 4: Autoscaling and Resource Management

Objective: Implement auto-scaling for the application to handle variable load, ensuring efficient resource utilization.

Activities:

- Horizontal Pod Autoscaler (HPA): Configure HPA for the application components to automatically scale the number of pods based on CPU utilization or custom metrics.

- Resource Requests and Limits: Define resource requests and limits in the application's deployment configurations to ensure that pods are scheduled on nodes with adequate resources and to prevent any single pod from monopolizing node resources.
- Cluster Autoscaler: If supported by the Kubernetes cluster environment, configure the Cluster Autoscaler to adjust the number of nodes in the cluster based on the demands of the deployed workloads.

Task 5: Monitoring, Logging, and Observability

Objective: Integrate monitoring, logging, and observability tools to maintain insight into application performance and health.

Activities:

- Monitoring Setup: Deploy a monitoring solution like Prometheus to collect metrics from the application and Kubernetes cluster. Use Grafana to create dashboards visualizing application performance.
- Logging: Implement a centralized logging solution using the EFK (Elasticsearch, Fluentd, Kibana) stack to aggregate and visualize logs from all application components.
- Distributed Tracing: Integrate a distributed tracing system such as Jaeger or Zipkin to trace requests across the microservices architecture, identifying bottlenecks and latency issues.

Deliverables:

- Dockerfiles and Docker Compose (if used for local testing) configurations for each application component.
- Kubernetes manifests for deploying the application, including networking and storage configurations.
- Documentation detailing the deployment process, configurations used, and reasoning behind design decisions.
- Monitoring, logging, and tracing configurations, along with dashboards and visualizations created for observing application performance.

==========

Project 2: Implementing a CI/CD Pipeline for Cloud-Native Applications

Project Overview :

As a business owner of a growing e-commerce platform, you recognize the need to accelerate feature releases while ensuring application reliability and security. Your application is hosted on Kubernetes, and you've decided to enhance your deployment process with an automated CI/CD pipeline. This pipeline must integrate security scanning to mitigate vulnerabilities and implement canary deployment strategies to gradually roll out changes, minimizing the risk to your production environment.

Objective:

Design and implement a CI/CD pipeline for your Kubernetes-hosted e-commerce application, incorporating automated security scans and canary deployment strategies to ensure fast, reliable, and secure delivery of software updates.

Task 1: CI/CD Pipeline Design and Tool Selection

Objective: Outline the CI/CD pipeline architecture and select the appropriate tools for implementation.

Activities:

- Determine the stages of the pipeline, including code integration, automated testing, security scanning, building, deploying, and post-deployment monitoring.
- Choose CI/CD tools (e.g., Jenkins, GitLab CI/CD, GitHub Actions, CircleCI) and security scanning tools (e.g., SonarQube, Clair, Snyk) that best fit the project's needs.

Task 2: Source Code Management and Integration

Objective: Set up source code management practices and configure the CI/CD pipeline for automated integration.

Activities:

- Implement a Git branching model (e.g., GitFlow or GitHub Flow) to manage the development lifecycle efficiently.
- Configure the CI tool to automatically trigger builds on code commits to the main repository, ensuring that every change is built and tested.

Task 3: Automated Testing and Security Scanning

Objective: Integrate automated testing and security scanning into the pipeline to identify issues early in the development cycle.

Activities:

- Develop unit and integration tests for the application components. Integrate these tests into the CI pipeline to run automatically on each build.
- Configure the security scanning tool to scan the codebase and dependencies for vulnerabilities. Fail the build if critical vulnerabilities are found, requiring them to be addressed before proceeding.

Task 4: Building Docker Images and Deploying to Kubernetes

Objective: Automate the process of building Docker images and deploying them to a Kubernetes environment.

Activities:

- Write Dockerfiles for the application components and configure the CI pipeline to build Docker images on successful code integration and testing.

- Use Kubernetes manifests or Helm charts to define the application deployment. Configure the CI/CD pipeline to update the Kubernetes environment with the new images, using a canary deployment strategy.

Task 5: Implementing Canary Deployments

Objective: Gradually roll out new features to a subset of users to ensure stability before full deployment.

Activities:

- Define canary deployment rules, such as routing a certain percentage of traffic to the new version of the application.
- Monitor key performance indicators (KPIs) and user feedback for the canary release. If issues arise, roll back to the previous version. Otherwise, gradually increase traffic to the new version until it serves all users.

Task 6: Monitoring and Feedback Loop

Objective: Set up monitoring for the deployed application and establish a feedback loop for continuous improvement.

Activities:

- Integrate monitoring tools (e.g., Prometheus, Grafana) to track application performance and user experience metrics in real-time.
- Use insights from monitoring and user feedback to guide further development priorities and pipeline improvements.

Deliverables:

- A detailed CI/CD pipeline configuration, including integration with source control, automated testing, security scanning, and deployment scripts.
- Documentation outlining the pipeline design, tool choices, testing and security strategies, and canary deployment process.
- Monitoring setup and dashboards for real-time application performance tracking.
- A rollback plan for handling deployment issues or critical vulnerabilities detected in production.

==========

Project 3: Stateful Application Management in Kubernetes

Project Overview :

Imagine you are the lead engineer for a software development team responsible for deploying and managing a blog platform that relies on a database to store posts and user comments. This platform is designed to be cloud-native and needs to be highly available and resilient to data loss. Your task is to deploy this stateful application on Kubernetes, ensuring that the database retains data across pod restarts and deployments and implementing a robust backup and recovery strategy.

Objective:

Deploy a blog platform as a stateful application on Kubernetes using StatefulSets and PersistentVolumeClaims (PVCs) to manage the database component. Additionally, demonstrate a backup and recovery process to ensure data durability and resilience.

Task 1: Preparing the Application and Environment

Objective: Prepare the blog platform's components for deployment, focusing on the stateful database component.

Activities:

- Containerize the blog platform's services, including the web frontend and the database backend, ensuring the database is configured to store data on a persistent volume.
- Create a Kubernetes cluster or use an existing one where the application will be deployed.

Task 2: Deploying with StatefulSets and PVCs

Objective: Utilize StatefulSets and PVCs for deploying the database to ensure persistent storage.

Activities:

- Create a StatefulSet for the database backend, ensuring that each pod has a stable hostname and that pods are created sequentially.
- Define PVCs to request persistent storage from Kubernetes, which will be mounted into the database pods, ensuring data is retained across pod restarts.

Task 3: Implementing a Backup and Recovery Strategy

Objective: Establish a process for backing up the database and recovering it in case of data loss.

Activities:

- Choose a backup tool or method suitable for the database technology being used (e.g., mysqldump for MySQL, pg_dump for PostgreSQL).
- Create a Kubernetes cron job that periodically executes the backup process, storing backup files in a secure and accessible location (e.g., cloud object storage).

- Document the procedure for restoring the database from a backup, including any necessary Kubernetes or database commands.

Task 4: Scaling and Managing the Stateful Application

Objective: Configure the application for scalability and high availability.

Activities:

- Implement horizontal scaling for the stateless components of the blog platform (e.g., web frontend) using Deployments and Horizontal Pod Autoscaler (HPA).
- Explore options for scaling the stateful component (database) if supported by the database technology, such as read replicas.
- Test the application's resilience by simulating failures and practicing the recovery process.

Task 5: Monitoring and Observability

Objective: Set up monitoring for the stateful application to ensure its health and performance.

Activities:

- Integrate monitoring tools like Prometheus and Grafana with the Kubernetes cluster to collect and visualize metrics from the blog platform.
- Create alerts for critical conditions, such as high CPU/memory usage on database pods, slow query response times, or failure to perform backups.

Deliverables:

- Dockerfiles for the blog platform's services and Kubernetes YAML manifests for deploying the application, including StatefulSets and PVCs.
- Scripts or instructions for performing database backups and recovery.
- Documentation detailing the deployment process, backup/recovery procedures, scaling strategies, and monitoring setup.
- A report on the application's resilience testing, including observed failures, recovery steps, and lessons learned.

==========

Project 4: Monitoring and Logging Setup for Kubernetes Applications

Project Overview :

As the lead architect for a software development team tasked with maintaining a high-traffic online education platform, you're responsible for ensuring the system's reliability and performance. The platform is hosted on Kubernetes, and you've noticed that visibility into the application's health and

performance could be improved. To address this, you've decided to implement a comprehensive monitoring and logging solution that provides real-time insights into both application behavior and infrastructure health, enabling proactive issue resolution and system optimization.

Objective:

Design and implement a robust monitoring and logging framework for a Kubernetes-hosted online education platform, utilizing open-source tools to create custom dashboards and configure alerts for key application and infrastructure metrics.

Task 1: Tool Selection and Setup

Objective: Choose and set up monitoring and logging tools suited for a Kubernetes environment.

Activities:

- Select monitoring tools (e.g., Prometheus for metric collection and Grafana for visualization) and a logging stack (e.g., Elasticsearch, Fluentd, and Kibana, also known as the EFK stack) that best fit the needs of the Kubernetes environment.
- Deploy the chosen tools on Kubernetes, ensuring they are configured to collect data from all relevant sources, including pod metrics, node health, and application logs.

Task 2: Configuring Log Collection and Management

Objective: Aggregate logs from across the Kubernetes cluster to facilitate centralized viewing and analysis.

Activities:

- Configure Fluentd as a log collector within the cluster, collecting logs from all pods and forwarding them to Elasticsearch for storage and indexing.
- Set up Elasticsearch indices and retention policies to manage log storage efficiently, ensuring logs are accessible for a defined period before being archived or deleted.

Task 3: Implementing Application and Infrastructure Monitoring

Objective: Set up Prometheus to scrape metrics from the Kubernetes cluster and application endpoints.

Activities:

- Deploy Prometheus with appropriate configurations to discover and scrape metrics from Kubernetes nodes, pods, and custom application metrics endpoints.
- Define alerting rules in Prometheus for critical conditions that require immediate attention, such as high memory usage, error rates, or pod failures.

Task 4: Dashboard Creation and Alert Configuration

Objective: Create Grafana dashboards for visualizing key metrics and configure alerts for proactive issue detection.

Activities:

- Design and create Grafana dashboards that provide insights into application performance, user activity, and system health. Dashboards should include metrics such as request latencies, error rates, system load, and pod resource utilization.
- Configure Grafana or Prometheus alerts to notify the development team via email or a messaging platform (e.g., Slack) when key metrics exceed threshold values, indicating potential issues.

Task 5: Backup, Recovery, and Documentation

Objective: Establish processes for backing up monitoring data and document the monitoring and logging setup.

Activities:

- Implement a backup strategy for Prometheus metric data and Elasticsearch indices to ensure monitoring data can be recovered in case of data loss.
- Document the monitoring and logging architecture, including tool configurations, data flow diagrams, dashboard guides, and alerting policies.
- Create a recovery guide detailing steps to restore monitoring and logging data from backups in case of failure.

Deliverables:

- Configuration files and deployment manifests for monitoring and logging tools on Kubernetes.
- A collection of Grafana dashboards providing comprehensive visibility into application and infrastructure metrics.
- Documentation covering the setup and configuration of the monitoring and logging stack, dashboard utilization, alert management, and backup/recovery procedures.
- ==========

Project 5: Network Policy and Security Hardening in Kubernetes

Project Overview :

As a Security Architect at a financial services company, you're tasked with enhancing the security posture of your company's Kubernetes-based payment processing application. Given the sensitive nature of financial transactions, it's crucial to ensure that the application is not only resilient against external threats but also protected from potential internal vulnerabilities. Your objective is to

implement network policies to regulate pod-to-pod communication within the Kubernetes cluster and harden the cluster's security by employing Role-Based Access Control (RBAC) and Pod Security Policies (PSP).

Objective:

Strengthen the security framework of a Kubernetes cluster hosting a critical payment processing application by configuring network policies for secure pod communication and applying RBAC and PSP for cluster security hardening.

Task 1: Assessment and Planning

Objective: Assess the current cluster configuration and plan the implementation of security measures.

Activities:

- Review the architecture of the payment processing application to understand the communication flow between microservices.
- Identify sensitive components of the application that require restricted access or special security considerations.

Task 2: Implementing Network Policies

Objective: Define and apply Kubernetes network policies to control the flow of traffic between pods within the cluster.

Activities:

- Create network policies that allow only necessary inter-service communications while blocking unwanted traffic between pods.
- Apply the network policies to the appropriate namespaces or pods, ensuring that the payment processing application's components can communicate as required but are isolated from other non-essential services.

Task 3: Role-Based Access Control (RBAC) Configuration

Objective: Set up RBAC to enforce least privilege access control for cluster resources.

Activities:

- Define roles with permissions tailored to the responsibilities of different users and services interacting with the Kubernetes cluster (e.g., developers, CI/CD tools, application services).
- Create role bindings to assign these roles to specific users, groups, and service accounts, ensuring that each entity has only the permissions necessary to perform its tasks.

Task 4: Pod Security Policies (PSP) Implementation

Objective: Deploy PSP to restrict the actions that pods can perform and prevent the execution of privileged operations.

Activities:

- Define PSPs that enforce security best practices, such as disallowing privileged containers, restricting host filesystem access, and limiting the use of volume types.
- Apply the PSPs to the cluster, associating them with the appropriate service accounts or user groups to ensure that the payment processing application runs securely without hindering functionality.

Task 5: Testing and Validation

Objective: Ensure that the implemented security measures function as intended without disrupting the application's operations.

Activities:

- Conduct tests to verify that network policies effectively isolate pod communications according to the defined rules.
- Test RBAC configurations by attempting to perform actions as different users and service accounts, verifying that access controls are correctly enforced.
- Validate that PSPs are applied and that pods are restricted according to the policies, attempting to deploy non-compliant pods to confirm they are blocked.

Task 6: Documentation and Knowledge Sharing

Objective: Document the security configurations and share knowledge with the team.

Activities:

- Create comprehensive documentation of the network policies, RBAC roles and role bindings, and PSP configurations, including the rationale behind each security measure.
- Prepare a presentation or workshop for the development and operations teams to explain the implemented security measures, their importance, and how to operate within the enhanced security framework.

Deliverables:

- Detailed network policy, RBAC, and PSP configurations tailored to the payment processing application's requirements.
- Test cases and results demonstrating the effectiveness and impact of the security measures on the application's functionality.
- Documentation and guides for maintaining and extending the security framework of the Kubernetes cluster.

- A knowledge-sharing session to ensure team members understand and can effectively work with the enhanced security measures.
- A report or presentation summarizing the observability features implemented, insights gained from monitoring and tracing data, and recommendations for improving application performance and reliability based on observability findings.