**Report on Coinbase Crypto Exchange Service Solution**

### 

#### **Table of Contents**

1. **Introduction**
2. **Solution Architecture**
3. **Deployment Architecture**
4. **CI/CD Pipeline Design**
5. **Security and Ethics Challenges**
6. **Implementation of Kubernetes Services**
7. **CI/CD Pipeline Implementation**
8. **Test Automation Suite**
9. **RunBook**
10. **Conclusion**

### **1. Introduction**

**This report provides an in-depth solution design for a container-based service to manage and monitor services related to the Coinbase crypto exchange. The design ensures high availability and resilience by leveraging Kubernetes as the orchestration platform and deploying services using a Blue-Green deployment model. A robust CI/CD pipeline is implemented to automate the deployment process. We also address the security and ethical challenges, focusing on secure data flows and compliance with industry standards.**

### **2. Solution Architecture**

**The solution architecture consists of various microservices to manage the core functions of the Coinbase exchange, such as buy/sell orders, user management, transaction history, and monitoring. These services are containerized using Docker and deployed in a Kubernetes cluster.**

**The architecture diagram illustrates:**

* **User Interaction via web or mobile UI.**
* **API Gateway to route requests to different services.**
* **Microservices for managing buy/sell operations and transactions.**
* **Cloud Database to store user data and transactional records.**
* **Monitoring and Logging services to track system health.**

**Data Flow:**

* **User requests flow through the API Gateway to the respective microservices.**
* **Microservices interact with the database to retrieve or store data.**
* **Monitoring services check the health and performance of each component.**

### **3. Deployment Architecture**

**The deployment architecture focuses on how the services are deployed within the Kubernetes cluster across multiple environments: Development (Dev), Testing (Test), and Production (Prod). We employ a Blue-Green deployment strategy to ensure zero downtime during service updates.**

**The diagram shows:**

* **Dev, Test, and Prod Environments.**
* **Blue-Green Deployment to switch between versions.**
* **Kubernetes Pods for each microservice, ensuring replication and scaling.**

**Request Flow:**

* **Blue and Green services run in parallel, and traffic can be switched between them seamlessly.**
* **Persistent storage is used for stateful services.**

### **4. CI/CD Pipeline Design**

**The CI/CD pipeline is designed to automate the build, test, and deployment process. It consists of:**

1. **Continuous Integration (CI): Code is automatically tested and built upon push to the repository.**
2. **Continuous Deployment (CD): Artifacts are automatically deployed to Dev, Test, and Prod environments using a Blue-Green deployment strategy.**

**The pipeline:**

* **Builds Docker images for each microservice.**
* **Pushes these images to AWS ECR.**
* **Deploys the containers into an AWS EKS Kubernetes cluster.**
* **Runs automated tests before promotion to production.**

### **5. Security and Ethics Challenges**

**Ensuring security is critical for any crypto exchange. The following measures were taken:**

1. **Data Encryption: All data at rest and in transit is encrypted using industry standards.**
2. **Authentication and Authorization: Services use OAuth 2.0 and JWT tokens for secure communication between services.**
3. **Security Audits and Monitoring: The system is monitored using tools like Prometheus and Grafana for security vulnerabilities.**
4. **Ethical Considerations: We adhere to best practices in handling sensitive user data, ensuring compliance with GDPR and CCPA standards.**

### **6. Implementation of Kubernetes Services**

**The core services are implemented as Kubernetes deployments:**

* **Buy/Sell Service: Handles user transactions and market orders.**
* **User Management Service: Manages user data, authentication, and authorization.**
* **Monitoring Service: Tracks system performance, logs, and errors.**

**Each service is defined in a YAML manifest file:**

**yaml**

**Copy code**

**apiVersion: apps/v1**

**kind: Deployment**

**metadata:**

**name: buy-sell-service**

**spec:**

**replicas: 3**

**template:**

**spec:**

**containers:**

**- name: buy-sell-container**

**image: my-repo/buy-sell-service:v1**

### **7. CI/CD Pipeline Implementation**

**For the CI/CD pipeline, GitHub Actions was chosen due to its seamless integration with GitHub. The pipeline configuration (.github/workflows/deploy.yml) is as follows:**

1. **Build Phase:**
   * **The pipeline builds Docker images from the source code.**
   * **These images are pushed to AWS ECR.**
2. **Test Phase:**
   * **Automated tests are run using Jest for unit testing.**
   * **Integration tests ensure the system is functioning correctly.**
3. **Deploy Phase:**
   * **Services are deployed to the EKS cluster in the respective environment (Dev, Test, Prod).**
   * **The Blue-Green deployment switches traffic between versions.**

### **8. Test Automation Suite**

**A simple test automation suite is built to run after each deployment, ensuring that critical functionality remains intact. Selenium is used for end-to-end testing, while Jest handles unit and integration tests.**

**Sample Test Case:  
javascript  
Copy code  
test('Buy order is successfully placed', async () => {**

**const response = await placeOrder({type: 'buy', amount: 100});**

**expect(response.status).toBe(200);**

**expect(response.data.success).toBe(true);**

**});**

**Tests are integrated into the CI/CD pipeline to ensure they run after every deployment.**

### **9. RunBook**

**The RunBook outlines the steps to deploy and manage the system:**

1. **Setup AWS EKS:**
   * **Use eksctl to create the Kubernetes cluster.**
   * **Configure kubectl to connect to the cluster.**
2. **Deploy Services:**

**Apply Kubernetes manifests for each service:  
bash  
Copy code  
kubectl apply -f buy-sell-service.yaml**

1. **Blue-Green Deployment:**
   * **Deploy the Blue environment first.**
   * **Switch traffic to the Green environment using Kubernetes services.**
2. **Monitor and Test:**
   * **Use Prometheus and Grafana to monitor service health.**
   * **Run automated test suites to validate deployments.**

### **10. Conclusion**

**This report provides a comprehensive solution for managing the Coinbase crypto exchange using a Kubernetes-based architecture. By leveraging Blue-Green deployment and a robust CI/CD pipeline, we ensure high availability, security, and automated testing, meeting Coinbase’s requirement of 100% uptime. The solution also addresses key security and ethical challenges, ensuring compliance with data protection standards.**

**This report should cover all aspects of the assignment in detail, showcasing a professional and high-quality solution for maximum marks.**

**4o**

##### **You said:**

**How to create RunBook?**

###### **ChatGPT said:**

**ChatGPT**

**A RunBook is a detailed, step-by-step guide that provides instructions for deploying, operating, and troubleshooting a specific system or solution. In the context of your Coinbase crypto exchange project, the RunBook will outline the steps needed to deploy and manage the Kubernetes-based solution, the CI/CD pipeline, and related automation processes.**

**Here’s how to create a RunBook for this assignment:**

### **1. Structure of the RunBook**

**Your RunBook should be organized clearly, with distinct sections that guide the reader through each stage of deployment and operation. The structure could be as follows:**

1. **Overview**
   * **Brief introduction to the system or service being managed.**
   * **Purpose of the RunBook.**
2. **Pre-requisites**
   * **Tools, accounts, and environments needed before starting the deployment process.**
3. **System Architecture Overview**
   * **Brief summary or link to the architecture diagrams (already in your report).**
4. **Step-by-Step Deployment**
   * **Detailed steps to deploy the service, including commands and configurations.**
5. **Operational Procedures**
   * **Ongoing tasks like monitoring, scaling, or restarting services.**
6. **Troubleshooting Guide**
   * **Common issues that might arise and how to resolve them.**
7. **CI/CD Pipeline Management**
   * **Steps to monitor and manage the pipeline (trigger builds, check statuses).**
8. **Monitoring and Alerts**
   * **Instructions for setting up or checking monitoring and alerting services.**

### **2. Creating Each Section of the RunBook**

#### **1. Overview**

**This section provides a brief description of the solution and why the RunBook is essential.**

**text**

**Copy code**

**The purpose of this RunBook is to guide users through the deployment, management, and troubleshooting of the Coinbase crypto exchange monitoring solution. This solution is deployed using Kubernetes on AWS EKS and managed through a CI/CD pipeline. The Blue-Green deployment method ensures 100% uptime while performing updates.**

#### **2. Pre-requisites**

**List everything needed before deploying the system.**

* **AWS Account: Ensure an AWS account is set up with sufficient permissions to deploy EKS clusters.**
* **Tools Installed:**
  + **Docker**
  + **kubectl (Kubernetes CLI)**
  + **eksctl (for creating an EKS cluster)**
  + **AWS CLI (to interact with AWS)**
  + **CI/CD Tool: GitHub Actions, Jenkins, or your selected tool**
* **Kubernetes Cluster: Ensure an EKS cluster is set up.**

**bash**

**Copy code**

**# Install kubectl**

**curl -LO "https://storage.googleapis.com/kubernetes-release/release/$(curl -s https://storage.googleapis.com/kubernetes-release/release/stable.txt)/bin/linux/amd64/kubectl"**

**chmod +x ./kubectl**

**sudo mv ./kubectl /usr/local/bin/kubectl**

**# Install eksctl**

**curl --silent --location "https://github.com/weaveworks/eksctl/releases/latest/download/eksctl\_$(uname -s)\_amd64.tar.gz" | tar xz -C /tmp**

**sudo mv /tmp/eksctl /usr/local/bin**

#### **3. System Architecture Overview**

**A high-level summary of the system, highlighting how services and data interact.**

**text**

**Copy code**

**The system is built on microservices architecture, with each service containerized using Docker. The system uses an API Gateway to route requests and interacts with cloud-native persistent storage (AWS RDS). The Kubernetes cluster manages scaling and deployment, while the Blue-Green deployment model ensures seamless upgrades.**

#### **4. Step-by-Step Deployment**

**Provide specific commands and steps to deploy the services to Kubernetes. This will include:**

* **Setting up the Kubernetes cluster.**
* **Deploying services.**
* **Managing the Blue-Green deployment process.**

**bash**

**Copy code**

**# Step 1: Create AWS EKS Cluster**

**eksctl create cluster --name coinbase-cluster --region us-west-2 --nodes 3**

**# Step 2: Configure kubectl to use the cluster**

**aws eks --region us-west-2 update-kubeconfig --name coinbase-cluster**

**# Step 3: Deploy the Buy-Sell Microservice**

**kubectl apply -f buy-sell-service.yaml**

**# Step 4: Verify the pods are running**

**kubectl get pods**

**# Step 5: Switch traffic to the new version using Blue-Green deployment**

**kubectl apply -f blue-green-deployment.yaml**

#### **5. Operational Procedures**

**Explain tasks like monitoring, scaling, or restarting services.**

**bash**

**Copy code**

**# Step 1: Monitoring Pods**

**kubectl get pods --namespace monitoring**

**# Step 2: Scaling Services (e.g., scale Buy-Sell service to 5 replicas)**

**kubectl scale deployment buy-sell-service --replicas=5**

**# Step 3: Restarting a Service**

**kubectl rollout restart deployment buy-sell-service**

#### **6. Troubleshooting Guide**

**Provide a list of common problems and solutions.**

* **Issue: Kubernetes pod fails to start.**
  + **Solution: Check logs using kubectl logs <pod\_name>.**
* **Issue: CI/CD pipeline fails.**
  + **Solution: Review the logs of the failed pipeline step. Re-run the pipeline if necessary.**

**bash**

**Copy code**

**# Step 1: Check service logs**

**kubectl logs <pod\_name>**

**# Step 2: Debug pod errors**

**kubectl describe pod <pod\_name>**

#### **7. CI/CD Pipeline Management**

**Outline how to interact with the CI/CD pipeline, trigger builds, and check the status of deployments.**

**text**

**Copy code**

**1. \*\*Triggering a Deployment\*\*: Push changes to the GitHub repository to automatically trigger the pipeline.**

**2. \*\*Monitoring Build Status\*\*: Go to the Actions tab (for GitHub Actions) or Jenkins dashboard to view the status of each pipeline run.**

**3. \*\*Blue-Green Deployment\*\*: Once the tests pass, the Blue environment is automatically promoted to Green, switching traffic seamlessly.**

#### **8. Monitoring and Alerts**

**Instruct how to monitor the system using tools like Prometheus and Grafana and set up alerts.**

**text**

**Copy code**

**1. \*\*Access Grafana\*\*: Visit the Grafana dashboard using the service URL.**

**2. \*\*Check Service Metrics\*\*: In the Grafana dashboard, monitor key metrics such as CPU usage, memory usage, and network traffic.**

**3. \*\*Set Up Alerts\*\*: Configure alerts in Grafana to notify the team if resource usage exceeds a predefined threshold.**

### **3. Final Tips for Your RunBook**

* **Be Detailed: Assume the person using the RunBook is not familiar with the system.**
* **Include Commands: Use real, tested commands wherever possible.**
* **Visuals: You can include screenshots or diagrams if it helps clarify steps.**
* **Version Control: Maintain a versioning section in the RunBook to track updates to the deployment process.**