**Comprehensive technical documentation for DevOps pipelines**

**1. Introduction**

**Purpose:** The purpose of this document is to provide comprehensive guidance on the DevOps path applied to the cloud-based system of Global Logistics. It divides the pipeline into its core components, and provides detailed guidance on setup, deployment, monitoring, and maintenance. By clearly explaining these processes, the document seeks to enable both technical and non-technical stakeholders to effectively understand, use and support the DevOps pipeline.

**Scope:** This document serves a wide audience, including project managers, developers, administrators and non-technical stakeholders. Each department is designed to provide valuable insights into pipeline operations. Technical readers will find step-by-step instructions, while non-technical stakeholders will gain an understanding of how the pipeline aligns with business objectives.

**DevOps pipeline overview:** The DevOps pipeline automates the process of integrating new code changes, accurately testing them, and deploying them in a production environment. Continuous integration (CI) automates the testing and integration of code changes, ensuring that new features are quickly validated. Continuous Deployment (CD) then automates the release of validated code to operational and production environments, reducing manual intervention and reducing the risk of errors.

**2. Set up a DevOps pipeline**

**Tools and techniques:**

**AWS Services:**

AWS EC2 provides scalable computing power, hosts our Jenkins server and supports instances for staging and production environments. AWS Elastic Load Balancing distributes incoming application traffic across EC2 instances, enhancing fault tolerance and reliability. This setup ensures optimal load distribution and uninterrupted service availability, which is essential to maintaining robust and efficient pipeline operations.

**Jenkins:** Jenkins is the cornerstone of the CI/CD pipeline. It coordinates the entire software creation and deployment process, and manages code compilation, testing, and deployment. Its plug-in system allows Jenkins to integrate with various version control systems, monitoring tools, and containerized transport platforms.

**Docker:** Docker enables applications to work continuously across different environments. Developers create container images that contain everything the app needs to run, ensuring the same behavior in development, test, and production environments.

**Git: Git** is a distributed version control system that tracks changes to source code. By linking it with Jenkins, developers can run automated builds and tests when code is committed in the repository.

**System Requirements:**

**Hardware requirements:** A Jenkins server requires a multi-core CPU with at least 16GB of RAM to handle concurrent versions. Docker hosts must be scalable, with enough CPU, memory, and storage space to run multiple containers simultaneously.

**Software requirements:** The Jenkins server must be running a stable version of Java, while Docker needs Linux with the latest kernel version. All systems must be patched and updated regularly.

**Configuration steps:**

**AWS Instances:** Three AWS Instances are set up to create robust infrastructure. The first example is for Jenkins, which acts as an automation server for continuous integration and deployment. The other two cases serve as staging and production environments, respectively. This separation ensures a structured pipeline that simulates real-world processes from testing to live deployment.

**Jenkins for Automation**: Jenkins was chosen for its powerful automation capabilities, which are essential for implementing continuous integration and deployment lines. By automating builds and deployments, Jenkins helps roll out new features and fixes quickly, which is essential to maintaining the high levels of service required by Global Logistics.

**Integration with GitHub**: By integrating Jenkins with GitHub, where web files are hosted, the system ensures that any code base changes are automatically pulled to Jenkins for build and testing. This integration supports a streamlined workflow where updates are continuously integrated and tested without manual intervention, enhancing the efficiency of the development team.

**Maven for Build and Test:** Using Maven within Jenkins to manage and create a software project simplifies the project creation process. This tool automatically handles project dependencies and ensures the build process is consistent across all environments.

**Docker for Consistency**: Docker is installed on both staging and production instances to containerize the application. This ensures that the application runs consistently across different compute environments, thereby reducing "working on my machine" errors and simplifying configuration management.

**3. Publication Procedures**

**Publishing workflow:**

**Code execution and build:** When developers submit code to the Git repository, Jenkins are configured to automatically detect those changes and run the build process. During this process, code is compiled, and several automated tests are performed to ensure the stability and functionality of the changes. This step is critical because it verifies that new additions to the code base integrate seamlessly with existing code, maintaining the integrity and reliability of the software.

**Staged deployment:** If automated tests pass without problems, Jenkins will continue to create a Docker image for the application. This image is then published in a phased environment that reflects the production environment as closely as possible. This stage is crucial for conducting integration testing and user acceptance testing in a controlled environment, helping to detect any potential issues before they reach production.

**Production deployment:** After successful validation in the staging environment, Jenkins automates application deployment on production servers. To reduce downtime and improve user experience during updates, use a blue-green deployment strategy. This strategy involves running two identical production environments that alternate as active and passive environments, allowing for a smooth transition and immediate rollback if necessary.

**Retraction Procedures:**

**Rollback to the previous version:** With Jenkins and Git, the system is configured to allow developers to revert to a specific stable version of the app if a problem arises after deployment. This rollback is performed by identifying and redeploying the last known good commit in the version control system, ensuring service continuity and system stability.

**Automatic alerts and crash recovery:** Integrated monitoring tools within Jenkins to continuously evaluate system performance after deployment. If they detect anomalies or performance drops, they trigger automatic alerts. Based on these alerts, the system can be configured to automatically revert to the previous stable version, ensuring that the application retains high availability and performance even in the face of deployment issues.

**4. Monitoring and maintenance**

**Monitoring tools:**

**خدمة AWS CloudWatch:**

CloudWatch acts as our vigilant guard. It monitors all of our AWS resources and applications , and collects data such as logs, metrics, and events to give us a comprehensive view of everything that happens in our AWS environment .

**-** It is especially useful for real-time monitoring capabilities, which allow us to track the health and performance of EC2 instances, Docker containers, and even the applications themselves. We use them to set up alarms and create dashboards that help us respond quickly to any changes or issues.

**Prometheus:**

Prometheus expands our monitoring capabilities, especially at the application level where details are key. It collects and stores its metrics as time series data, allowing us to query and analyze this data to understand deeper trends.

**-** This tool is ideal for tracking detailed operational metrics and is often paired with Grafana to visualize these metrics effectively. This combination helps us detect and address issues before they affect our services.

**Grafana:**

Grafana takes the data collected by Prometheus and transforms them into insightful visualizations. This makes it easier for us to ingest complex data and make informed decisions quickly.

-It's all about clarity and actionable insights. Grafana dashboards are very valuable to our team, helping us monitor everything from server loads to application response times in a visually intuitive format.

**Monitoring metrics:**

**System uptime:** Constant monitoring of system uptime is critical to ensure that the application is always available to users. Downtime is closely tracked, with targets set to achieve and maintain uptime of 99.9% or higher, reflecting system reliability.

**Error rates:** Monitoring the frequency and type of application errors helps in the early detection of issues that are likely to escalate into more significant issues. By identifying and addressing these errors quickly, the system maintains higher levels of operational efficiency and user satisfaction.

**Network traffic and latency:** By monitoring the amount of data flowing in and out of our systems and how long it takes to process requests, we can detect potential network issues early. It is crucial to maintain a fast and reliable service for our users.

**Resource usage:** It is essential to continuously monitor CPU, memory, and disk usage to detect and address potential bottlenecks. Efficient use of resources helps improve performance and can prevent system overload, ensuring that the application runs smoothly at all times.

**Maintenance Procedures:**

**Regular updates:** To protect the app and its platforms from vulnerabilities, regular updates are required for all tools, libraries, and dependencies. These updates are scheduled during periods of low traffic to minimize the impact on system availability.

**Backups and recovery**: Regular and systematic backup of all important data and configuration files is essential for disaster recovery. These backups are periodically tested to ensure that they can be restored quickly and effectively, which is vital for business continuity.

**Security audits:**  Routine security audits are performed to identify and correct any vulnerabilities within the system. These audits help maintain the integrity and security of the app and user data, in line with industry best practices and compliance requirements.

**5. Troubleshooting guide**

**Common issues and decisions:**

**Build failure:** Detailed logging and error reporting are used to help trace the root cause of build failure back to specific code changes or configuration errors. This section provides a step-by-step approach to diagnosing and resolving construction issues, which is critical to maintaining a smooth and efficient development path.

**Deployment errors:** This part of the guide discusses common scenarios in which deployments fail due to misconfigurations or resource constraints. It includes inspections and modifications for environment variables, resource allocations, and dependency conflicts to ensure successful deployments.

**Support contact information:** Provides a comprehensive list of contact details for the internal technical support team, including direct phone numbers, email addresses, and appropriate times to contact support. This section also explains the escalation process for different severity levels, ensuring critical issues are quickly escalated to the right experts.

**6. Appendix**

This section outlines the technical terms and acronyms used throughout the document, such as CI/CD, Docker file, and undo, ensuring that all stakeholders have a clear understanding of the terminology.

Links to extensive resources are included, such as official documents for Jenkins, Docker,Git, Prometheus, andGrafana. This section also includes internal best practice guides and security policies, providing users with access to a wealth of information for deeper understanding and effective use of the DevOps pipeline.

This extensive documentation provides a comprehensive and detailed guide to managing and understanding the DevOps pipeline in Global Logistics, ensuring clarity and efficiency in ongoing operation and maintenance.