Crawler **Design Doc**

*Authors:* [*aqif.cse@gmail.com*](mailto:aqif.cse@gmail.com) *April, 2024*

# **One-page overview**

Here is a proposed solution to create a more scalable, cost-effective, and maintainable web crawler and infrastructure solution that can handle the crawling of 150+ sites, including dynamic addition of new sites, while minimizing operational overhead and costs.

### **Summary**

The proposed solution for the web crawler system involves modular components for fetching, parsing, and storing data, enabling easy scalability and maintainability. By utilizing containerization, orchestration with Kubernetes, and serverless computing, the infrastructure can be optimized for cost efficiency and seamless management. Continuous integration and deployment pipelines ensure automated testing and deployment, streamlining development workflows and enhancing reliability.

### **Platforms**

Linux, GCP

### **Team**

SYED TOWFIQUR RAHIM ([aqif.cse@gmail.com](mailto:aqif.cse@gmail.com))

### **Bug**

### **Code affected**

# **Design**

**1. Redesigning the Web Crawler:**

**Modular Architecture:**

* Design the web crawler with a modular architecture where components responsible for fetching web pages, parsing data, and storing information are decoupled.
* Create reusable components for handling HTTP requests, HTML parsing, data extraction, and storage.

**Configurable Site Definitions:**

* Develop a configuration-based approach where each site to be crawled is defined in a configuration file.
* Include parameters such as URL patterns to crawl, data extraction rules, frequency of crawling, and any site-specific configurations.

**Dynamic Site Addition:**

* Implement a mechanism to dynamically add new sites to crawl without modifying the core crawler code.
* New sites can be added by updating the configuration files, eliminating the need for code changes.

**Error Handling and Monitoring:**

* Incorporate robust error handling mechanisms to handle exceptions during crawling gracefully.
* Integrate logging functionalities to track the status of each crawling process and record any errors encountered.
* Utilize monitoring tools like Prometheus and Grafana to monitor the health and performance of the crawler.

**2. Infrastructure Optimization:**

**Containerization:**

* Containerize the web crawler using technologies like Docker to create lightweight, portable, and scalable deployment units.
* Each site's crawler can run within its own container, ensuring isolation and easier management.

**Orchestration with Kubernetes:**

* Utilize Kubernetes for container orchestration to manage and scale the crawler instances efficiently.
* Kubernetes can handle scheduling, scaling, and monitoring of containers across a cluster of nodes.

**Serverless Approach:**

* Consider leveraging serverless computing platforms like AWS Lambda or Google Cloud Functions for executing the crawler code.
* Serverless architecture eliminates the need to manage infrastructure and automatically scales based on demand, reducing costs.

**Centralized Logging and Monitoring:**

* Implement centralized logging solutions such as ELK stack (Elasticsearch, Logstash, Kibana) or Google Cloud Logging to aggregate logs from all crawler instances.
* Use monitoring tools like Prometheus and Grafana for real-time monitoring of resource usage, performance metrics, and error tracking.

**3. Continuous Integration and Deployment (CI/CD):**

**Automated Testing:**

* Implement automated testing for the crawler codebase to ensure reliability and consistency.
* Include unit tests, integration tests, and end-to-end tests to validate the functionality of the crawler.

**Continuous Integration:**

* Set up a CI/CD pipeline using tools like Jenkins or GitLab CI to automate the build, test, and deployment processes.
* Any changes to the crawler code are automatically tested and deployed to the production environment, ensuring quick iterations and minimizing downtime.

**4. Cost Optimization:**

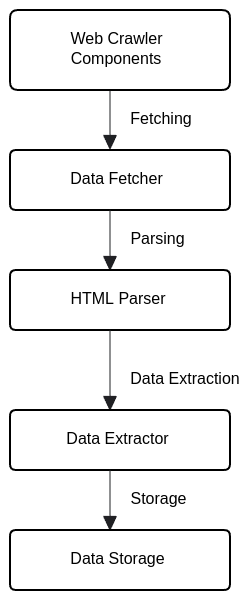
**Reserved Instances or Spot Instances:**

* Utilize reserved instances or spot instances on cloud providers like AWS or Google Cloud to reduce costs for long-running instances.
* Reserved instances offer significant discounts for predictable workloads, while spot instances can be used for non-critical tasks at a lower cost.

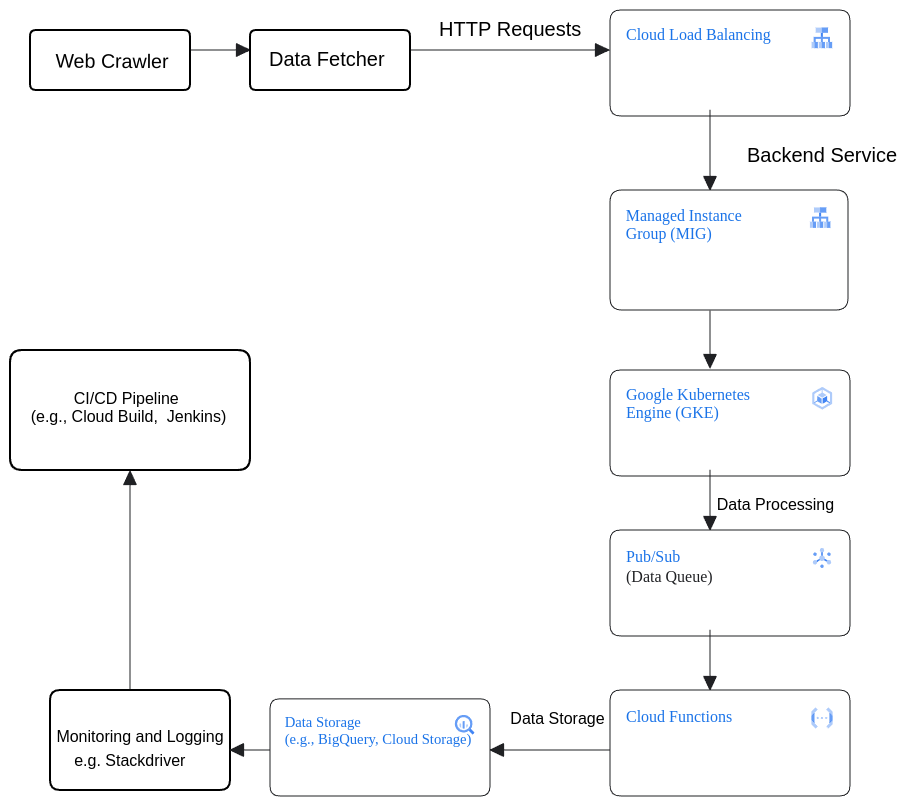
**Autoscaling:**

* Implement autoscaling policies based on resource utilization metrics to dynamically adjust the number of crawler instances.
* Autoscaling ensures optimal resource allocation and cost efficiency by scaling up or down based on demand.

**Generic Crawler System Architecture Diagram:**

****

**Cloud Implementation steps**

****

**Here is the redesigned architecture**

* The "Web Crawler" component remains at the top, representing the overall system responsible for crawling web data.
* The "Data Fetcher" component fetches data from web pages, initiating HTTP requests to retrieve content.
* The "Google Cloud Load Balancer (GCLB)" ensures even distribution of incoming HTTP traffic across backend services.
* The "Managed Instance Group (MIG)" hosts the web crawler application, managing and scaling instances as needed.
* The "Google Kubernetes Engine (GKE)" provides a managed Kubernetes environment for orchestrating containerized applications, including the web crawler.
* "Pub/Sub (Data Queue)" serves as a messaging queue for distributing tasks related to data processing and handling.
* "Google Cloud Functions" executes serverless functions in response to events, performing specific tasks or processing data related to web crawling.
* "BigQuery (Data Warehouse)" stores crawled data for analysis and querying.
* "Monitoring & Logging" collects and analyzes system logs and metrics, ensuring visibility into system health and performance.
* "CI/CD Pipeline" automates the build, test, and deployment processes, ensuring continuous integration and delivery of code changes.

# Metrics

## Success metrics

1. **Crawl Success Rate:** Measure the percentage of sites successfully crawled compared to the total number of sites scheduled for crawling. A high crawl success rate indicates the effectiveness of the crawler in fetching data from diverse sites.
2. **Data Extraction Accuracy**: Evaluate the accuracy of data extraction by comparing the extracted data with the expected data from sample sites. Higher accuracy ensures reliable data for downstream processes.
3. **Infrastructure Cost Reduction:** Monitor the reduction in infrastructure costs achieved through optimization techniques such as containerization, serverless computing, and efficient resource allocation.
4. **Scalability:** Measure the ability of the system to handle an increasing number of sites without significant degradation in performance or increase in resource consumption.

## Regression metrics

1. **Error Rate:** Track the rate of errors encountered during crawling, parsing, data extraction, and storage. An increase in error rate may indicate regressions or issues introduced during code changes or updates.
2. **Performance Metrics:** Monitor key performance indicators such as response time, throughput, and resource utilization to detect any regressions in system performance.
3. **Data Consistency:** Ensure consistency in the extracted data across different runs of the crawler. Any deviation from expected data patterns could signify regressions in data extraction logic.

## Experiments

1. **Containerization vs. Serverless:** Conduct experiments to compare the performance, scalability, and cost-effectiveness of containerized deployment versus serverless deployment for the web crawler system.
2. **Optimization Techniques:** Experiment with different optimization techniques such as caching mechanisms, parallel processing, and load balancing to improve crawler efficiency and reduce resource utilization.
3. **Error Handling Strategies:** Evaluate different error handling strategies such as retry mechanisms, fault tolerance, and graceful degradation to identify the most effective approach for handling errors during crawling.
4. **Scaling Strategies:** Test various scaling strategies, including vertical scaling (increasing resources on existing instances) and horizontal scaling (adding more instances), to determine the most efficient method for scaling the crawler based on workload fluctuations.
5. **Continuous Integration and Deployment:** Experiment with different CI/CD pipelines configurations, testing suites, and deployment strategies to optimize the development workflow and ensure smooth deployments with minimal downtime.

# **Rollout plan**

The rollout plan for the web crawler system entails a phased approach, starting with development, followed by beta testing, and culminating in a stable release. Each stage involves iterative development, testing, and refinement to ensure the system's robustness and reliability. Beta testing will involve deploying the system to a limited environment to gather feedback and address any identified issues. Once stable, the system will be released to production, with ongoing monitoring and optimization to maintain performance. Experimentation will be conducted to explore optimization techniques, error handling strategies, and CI/CD pipeline configurations. Variations of the system will be deployed to assess the impact of different approaches on performance, scalability, and cost-efficiency. Through continuous iteration and improvement, the rollout aims to deliver a highly effective and efficient web crawler system.

**Core principle considerations:**

**Speed:**

* The rollout plan prioritizes optimizing the speed and efficiency of the web crawler system, ensuring minimal impact on performance metrics such as response time and resource consumption.
* Performance impacts will be monitored through benchmarks and metrics specific to the crawling process, with emphasis on maintaining or improving speed launch metrics.
* Experimentation with optimization techniques will be conducted to identify and implement strategies for enhancing speed without compromising reliability.

**Simplicity:**

* User-facing changes in the web crawler system will be designed to minimize complexity and cognitive load, focusing on intuitive interfaces and streamlined workflows for users managing crawling tasks.
* Consideration will be given to minimizing the need for users to make additional decisions or understand new concepts, aiming to reduce switching costs and enhance user adoption.
* Collaboration with stakeholders and user feedback loops will guide decisions to ensure that changes align with simplicity principles and user needs.

**Security:**

* A comprehensive threat model will be developed to identify potential security risks and vulnerabilities in the web crawler system, with a focus on handling untrusted data securely.
* Security mechanisms such as input validation, data sanitization, and access controls will be implemented to mitigate identified risks and protect against potential points of failure.

**Privacy considerations:**

* Features of the web crawler system with privacy implications will undergo thorough privacy reviews to assess and mitigate any potential privacy risks.
* Privacy-preserving measures, such as anonymization of data and encryption of sensitive information, will be implemented to safeguard user privacy and comply with relevant privacy regulations.

**Testing plan:**

* The testing plan will focus on ensuring the reliability, robustness, and security of the web crawler system through comprehensive testing methodologies.
* Testing will encompass unit tests, integration tests, and end-to-end tests to validate the functionality of individual components and the system as a whole.
* Special attention will be given to testing performance under various conditions to ensure that speed and efficiency targets are met.

**Follow-up work:**

* The success of the rollout will be evaluated based on predefined success metrics, including crawl success rate, data extraction accuracy, and system performance.
* Follow-up activities may include addressing any identified issues or enhancements, optimizing performance based on feedback and monitoring, and refining security measures to mitigate emerging threats.