**BoltQ (Distributed Task Queue) - Project Documentation**

**Overview**

BoltQ is a **Distributed Task Queue (DTQ)**, a **scalable, event-driven job processing system** built using **Golang** and **Redis**. It enables **asynchronous execution of tasks** by decoupling job submission from execution, ensuring efficient resource utilization and fault tolerance.

The system consists of multiple components, including an **API service**, **task queue**, and **worker nodes** that process tasks. This setup allows tasks to be distributed across multiple worker nodes, ensuring load balancing and scalability.

**Key Features**

* **Job Submission API**: Exposes endpoints for submitting and querying tasks.
* **Task Queue**: Stores and distributes jobs to worker nodes.
* **Worker Nodes**: Process tasks asynchronously.
* **Scalability**: Add more worker nodes as demand increases.
* **Fault Tolerance**: Failed jobs are retried or sent to a dead-letter queue.
* **Observability**: Logs and metrics to monitor job processing.
* **Pluggable Queue Backend**: Start with Redis, but extendable to Kafka or NATS.

**Architecture**

**High-Level Flow**

1. **Client submits a job** via the API.
2. The API **pushes the job into Redis Streams**.
3. **Workers consume jobs** from Redis and process them asynchronously.
4. Job results can be stored in a **database** or sent to **a callback URL**.
5. Logs and monitoring tools track **job status and performance**.

**Components**

| **Component** | **Description** |
| --- | --- |
| **Job API** | HTTP API for job submission and status checking. |
| **Redis Queue** | Stores tasks and enables message passing between API and workers. |
| **Worker Service** | Consumes tasks from the queue and executes them. |
| **Logger** | Handles structured logging and error tracking. |
| **Config Manager** | Manages environment variables and service configuration. |

**Phased Development Plan**

To implement the DTQ efficiently, we will break down the development into multiple phases. Each phase will progressively add new functionality and improvements.

**Phase 1: Basic Job Submission and Processing**

* Set up the **Job API** to accept job submissions.
* Implement **Redis Queue** for task storage.
* Create a **basic worker service** that picks jobs from the queue and executes them.

**Phase 2: Job Status Tracking and Error Handling**

* Implement job **status tracking** in Redis.
* Add **error handling** and retries for failed jobs.
* Introduce a **dead-letter queue** for failed jobs.

**Phase 3: Scaling and Optimization**

* Support **multiple worker nodes** to scale processing.
* Implement **load balancing** mechanisms.
* Optimize **job queue management** for performance.

**Phase 4: Observability and Monitoring**

* Add **logging and monitoring tools** for job execution tracking.
* Expose **metrics** (e.g., number of jobs processed, failed jobs, etc.).
* Implement **distributed tracing**.

**Phase 5: Advanced Features and Extensions**

* Support **priority queues** to handle high-priority tasks first.
* Add support for **delayed job execution**.
* Extend the queue backend to support **Kafka or NATS**.
* Implement **dashboard UI** for job monitoring.

**Phase 6: Frontend Playground for Users**

* Develop a **web-based frontend** to interact with the system.
* Provide a **visual playground** for users to submit jobs and see live updates.
* Implement **real-time job tracking** and monitoring.
* Deploy the frontend alongside the backend to offer a complete hosted experience.

**Next Steps**

We will now proceed with **Phase 1**, setting up the core system:

* Implementing the **Job API**.
* Setting up **Redis Queue**.
* Creating a **Basic Worker Service**.

After Phase 1 is complete, we will incrementally add features following the planned phases.

Let me know if you have any modifications or additional requirements! 🚀

Project Directory:  
  
📄 .gitignore

📄 Dockerfile

📄 Makefile

📄 README.md

📁 cmd/

📁 api/

📄 main.go

📁 worker/

📄 main.go

📄 go.mod

📁 internal/

📁 api/

📄 handler.go

📁 queue/

📄 redis\_queue.go

📁 worker/

📄 worker.go

📁 pkg/

📁 config/

📄 config.go

📁 logger/

📄 logger.go