Igor Komlew

Engineering Case (Cat) Study

```
 ,_     _
 |\\_,-~/
 / _  _ |    ,--.
(  @  @ )   / ,-'
 \  _T_/-._( (
 /         `. \
|         _  \ |
 \ \ ,  /      |
  || |-_\__   /
 ((_/`(____,-'
```

Content

I. Actions to Address Current Problems

II. Practices and Process

III. Migration and Architecture

Actions to Address Current Problems

AKA Roadmap

Performance and Scalability

- Implement horizontal scaling by introducing load balancers and multiple application server instances to handle spikes in traffic.
- Utilize caching mechanisms (e.g., Redis or Memcached) to reduce the load on the database and improve response times.
- Scale the database by introducing replication.
- Explore the possibility of using a Content Delivery Network (CDN) to distribute static assets globally and reduce latency.

Stability and Monitoring

- Implement comprehensive monitoring and alerting systems to detect and address issues proactively.
- Set up automated health checks and proactive monitoring of critical components.
- Implement centralized logging to collect and analyze logs for troubleshooting and identifying patterns.

Quality Assurance

- Establish a dedicated testing environment with comprehensive test suites, including unit tests, integration tests, and end-to-end tests.
- Implement continuous integration and deployment (CI/CD) pipelines to automate the testing and release processes.
- Introduce code reviews and pair programming practices to ensure higher code quality and catch issues early.

Bug Detection and Prioritization

- Implement error tracking and bug reporting tools to capture and prioritize issues reported by customers.
- Establish a process for triaging and prioritizing bugs based on their impact on customers and the business.
- Introduce proactive bug hunting by conducting regular code reviews, static code analysis, and automated vulnerability scanning.

Security and Compliance

- Perform a thorough security audit to identify and address potential vulnerabilities and compliance gaps.
- Implement secure coding practices, including input validation, parameterized queries, and output encoding, to mitigate common security risks.
- Introduce data encryption at rest and in transit to protect customer data.
- Establish a clear data privacy policy and ensure compliance with relevant regulations (e.g., GDPR).

Practices and Process

Mentoring and Knowledge Sharing

- Encourage senior developers to mentor junior peers and facilitate knowledge sharing sessions.
- Organize regular code reviews and pair programming sessions to provide guidance and foster learning opportunities.
- Implement a culture of continuous learning by providing resources, training programs, and encouraging participation in relevant conferences and workshops.

Training and Skill Development

- Identify skill gaps among junior developers and create personalized training plans to address those gaps.
- Invest in training programs, workshops, and online courses to enhance technical skills and introduce modern software engineering practices.
- Encourage junior developers to participate in challenging projects and provide opportunities for growth and career progression.

Pair Programming and Peer Review

- Promote pair programming practices where junior developers can work alongside experienced developers, learn best practices, and receive immediate feedback.
- Establish a culture of code reviews, ensuring that all code changes are reviewed by senior developers to maintain high code quality standards.
- Hackaton.

Agile Practices

- Introduce agile methodologies (e.g., Scrum or Kanban) to improve collaboration and efficiency within the development team.
- Implement regular retrospective meetings to identify areas for improvement and address any bottlenecks or challenges.
- Foster a supportive environment where questions and ideas are encouraged, and constructive feedback is provided.

Target Architecture 1

- Transition from a monolithic application to a microservices architecture, allowing independent development, deployment, and scalability of individual services.
- Utilize containerization technologies like Docker and orchestration platforms like Kubernetes for efficient service deployment and management.

Target Architecture 2

- Introduce event-driven architecture and asynchronous communication patterns to decouple services and improve responsiveness (if needed).
- Implement API gateways and service meshes to handle authentication, rate limiting, and traffic management.

Force multiplier

- Continuous Learning and Improvement
- Implement Best Practices
- Invest in Tools and Infrastructure
- Cross-Functional Collaboration
- QA and Automation
- Monitor and Analyze Performance
- Innovation and Experimentation

Migration

Analyze and Decompose

- Understand the existing monolithic application's functionality, dependencies, and internal components.
- Identify logical boundaries and potential microservices based on domain-driven design principles.
- Define the communication interfaces and API contracts between the microservices.

Identify Core Functionality

- Document the current state of the application.
- Determine the critical features and functionality that should be prioritized for migration.
- Start with smaller, less complex modules that can be extracted and rewritten as independent microservices.

Design the Architecture

- Define the target architecture for your microservices, including service boundaries, communication patterns, and data storage requirements.
- Separate frontend and the backend parts of the application.
- Consider using existing tools/frameworks as much as possible to simplify building microservices in Go and frontend part in React.
- Establish a service discovery mechanism for the microservices to communicate with each other (DNS, Service Registry).

Assess Team and Resources

- Evaluate the skill sets and expertise of your existing development team.
- Identify team members who have experience or interest in learning Go and working with microservices or frontend and working with React.
- Determine if additional training or hiring of external Go/React developers and devops is required to supplement the team's capabilities.
- Assess the available resources, including budget, time constraints, and infrastructure capacity.

Define the Migration Strategy

- Determine which parts of the migration can be effectively handled by the existing team and which parts might benefit from external expertise.
- Consider outsourcing specific tasks or components to external agencies or contractors to expedite the process and ensure a smooth transition.
- Clearly define the scope, deliverables, and timelines for the outsourced work and establish effective communication channels with the external partners.

Develop and Test

- Begin building the microservices in Go and frontend in React, starting with the identified core functionality.
- Follow best practices for microservice development, such as single responsibility, loose coupling, and proper error handling.
- Write unit tests and integration tests to ensure the correctness of each microservice.

Implement Communication I

- Choose a communication protocol, such as HTTP/REST or gRPC, for inter-service communication.
- Develop APIs and define contracts for communication between microservices.
- Integrate a service mesh or API gateway to handle common cross-cutting concerns like authentication, rate limiting, and monitoring.

Implement Communication II

- Think about the communication with the frontend components.
- Add backend for frontend (BFF) or GraphQL if needed.
- Design a public API with the possibility to be consumed not only by UI but also by mobile apps and developers.

Data Management and Persistence

- Decide on the data storage strategy for each microservice.
- Consider using separate databases for each microservice or a shared database with strict data isolation.
- Implement data access layers within each microservice to handle database interactions.

Deploy and Orchestrate

- Containerize each microservice using Docker and create container images.
- Utilize container orchestration platforms like Kubernetes to manage the deployment, scaling, and monitoring of microservices.
- Establish a CI/CD pipeline to automate the build, test, and deployment process.

Monitor, Measure, and Optimize

- Implement monitoring and logging solutions to gain insights into the performance and health of the microservices.
- Define key performance indicators (KPIs) to measure the effectiveness and efficiency of the new architecture.
- Continuously optimize the microservices by identifying bottlenecks, improving resource utilization, and optimizing the codebase.

Gradual Migration

- Plan for a phased migration approach where you gradually replace modules of the monolith with corresponding microservices.
- Maintain backward compatibility during the migration to minimize disruption for end-users.
- Gradually retire the monolithic codebase as more microservices become operational.

Monitor and Evaluate

- Continuously monitor the performance, stability, and scalability of the microservices architecture.
- Gather feedback from developers and end-users to identify areas for improvement.
- Iterate and refine the architecture based on the evolving needs of the application.

Thanks!

Questions?