

Operation am offenen Herzen: Erfahrungen einer nahtlosen zentralen API Migration

Jan Nonnen, Engineering Manager, SAP LeanIX
FrOSCon 2024

Agenda

1. Überblick SAP LeanIX
2. Ausgangslage
3. Wie sind wir vorgegangen
4. Zusammenfassung

Über mich

- Engineering Manager bei SAP LeanIX
 - Team verantwortlich für Modernisierung und Weiterentwicklung der Suche
- Full-Stack Entwickler und Scrum Master
- Frisch gewordener Vater im März
- Wohnt in Königswinter - Stieldorf

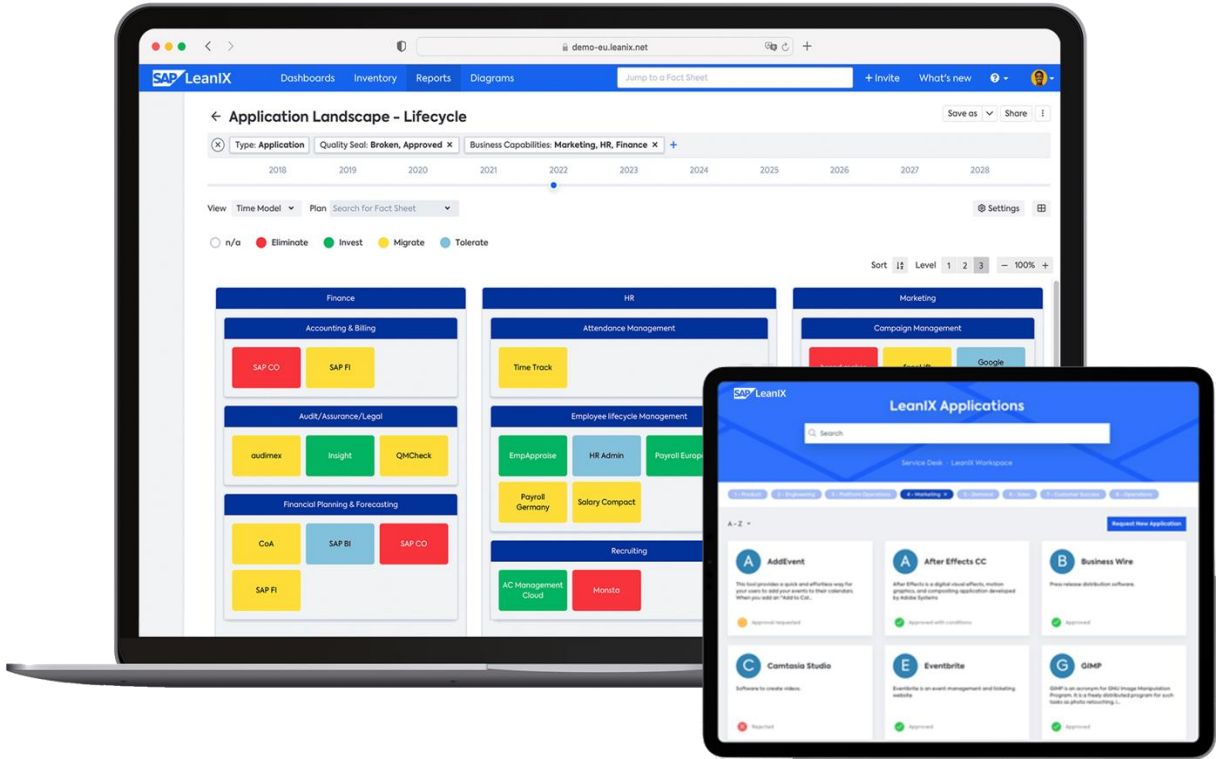
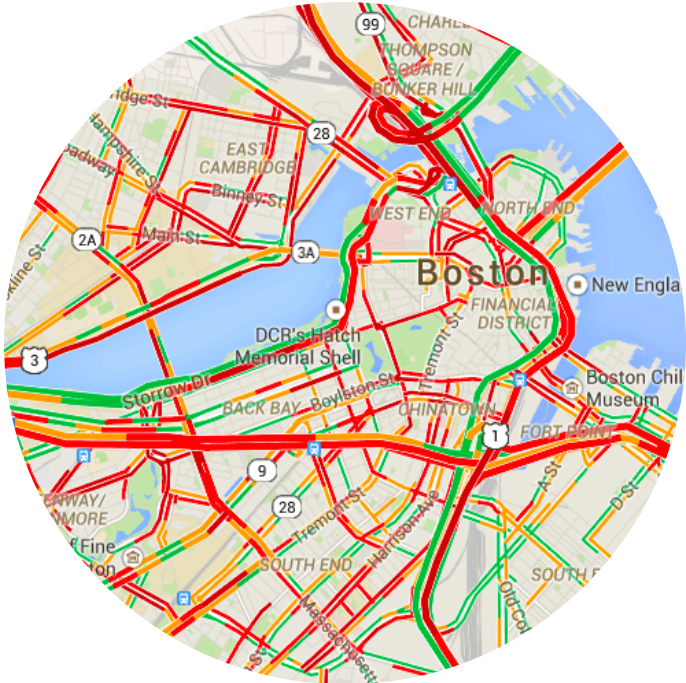


Überblick SAP LeanIX

- Gegründet 2012 mit Sitz in Bonn
- Gekauft von SAP Oktober 2023
- 700+ Mitarbeiter in 23 Ländern
- 200+ Entwickler
- Software as a Service mit 1300+ Kunden



SAP LeanIX to navigate your transformation



adidas

DHL

BOSCH

Lufthansa

OTIS

MIGROS

salesforce

MCKESSON

+ 1,300 more

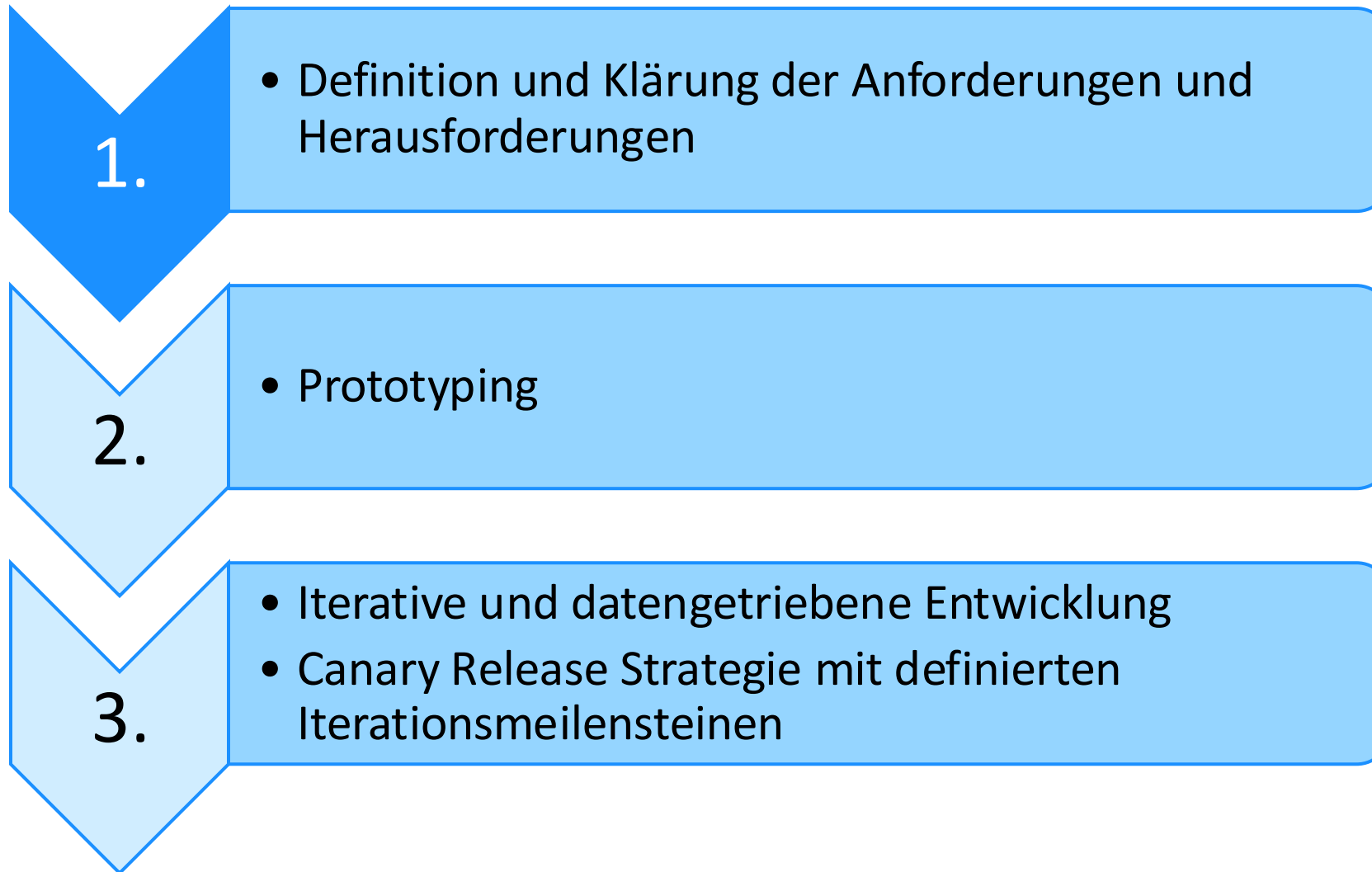
Ausgangslage – GraphQL Modernisierung und Extraktion

- Langfristiges Architektur Ziel
 - Extraktion von Funktionalität aus dem Monolithen in Microservices
- GraphQL ist unsere wichtigste Kunden-API
 - Schema-basierte API auf Basis von REST
 - Jeder Kunde hat ein eigenes Modell und dadurch unterschiedliches Schema
 - Kunden haben Skripte die auf existierende API und URL gehen
 - ~10 Millionen tägliche API Aufrufe von externen Systemen anfangs (~13 Mio aktuell)
 - Ein Aufruf kann tausende GraphQL Queries/Mutationen enthalten

Ausgangslage – GraphQL Modernisierung und Extraktion

- Technische Herausforderungen:
 - Alter GraphQL Server Code ist eine Kopie auf Stand von 2018
 - Wenige Entwickler noch vorhanden die 2018 an dem Code gearbeitet haben
 - GraphQL Schema ist nicht standard-compliant
 - Kein Monitoring möglich über detaillierte Verwendung von Feldern / Operationen
 - Nicht alle weiteren Services sollen GraphQL können müssen
 - Monolith ist in VMs, neue Microservices sind in Kubernetes mit anderem Netzwerkrouting

Wie sind wir vorgegangen



Vier große Produktrisiken nach Marty Cagan

Value Risk: Änderungen haben Auswirkungen auf bestehende Integrationen der Kunden

Usability Risk: Änderungen in der Architektur und der API könnten die Benutzerfreundlichkeit beeinträchtigen

Feasibility Risk: Technische Machbarkeit und Herausforderungen für Skalierbarkeit & Resilienz

Business Viability Risk: Beeinträchtigung der Stabilität und Performanz der API insbesondere mit Änderungen der Architektur

=> Fokus auf Feasibility Prototypen

PRODUCT TEAM LESSONS FROM
THE WORLD'S TOP TECH COMPANIES

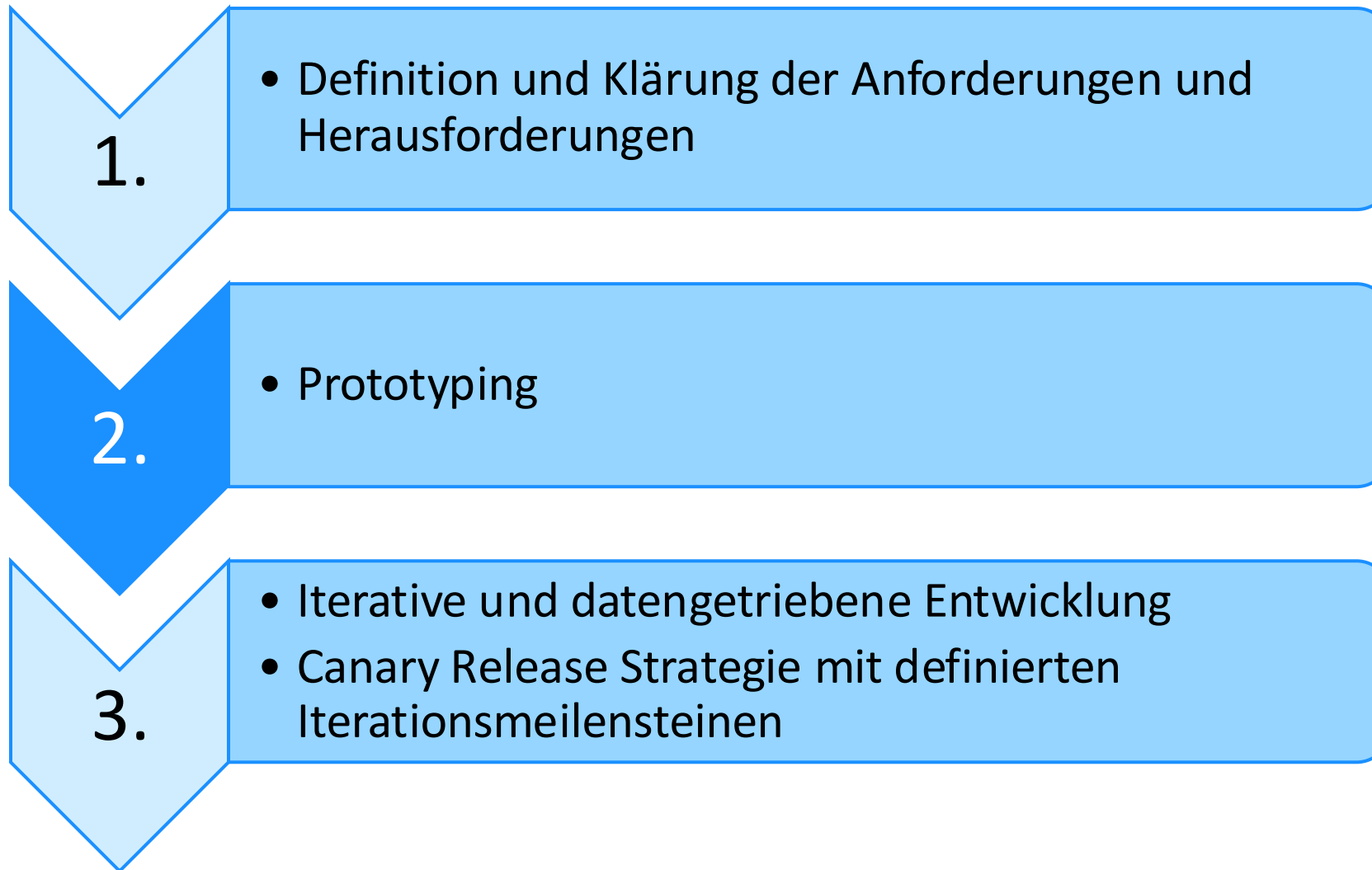
MARTY CAGAN
Silicon Valley Product Group

INSPIRED

HOW TO
CREATE
TECH
PRODUCTS
CUSTOMERS
LOVE

WILEY

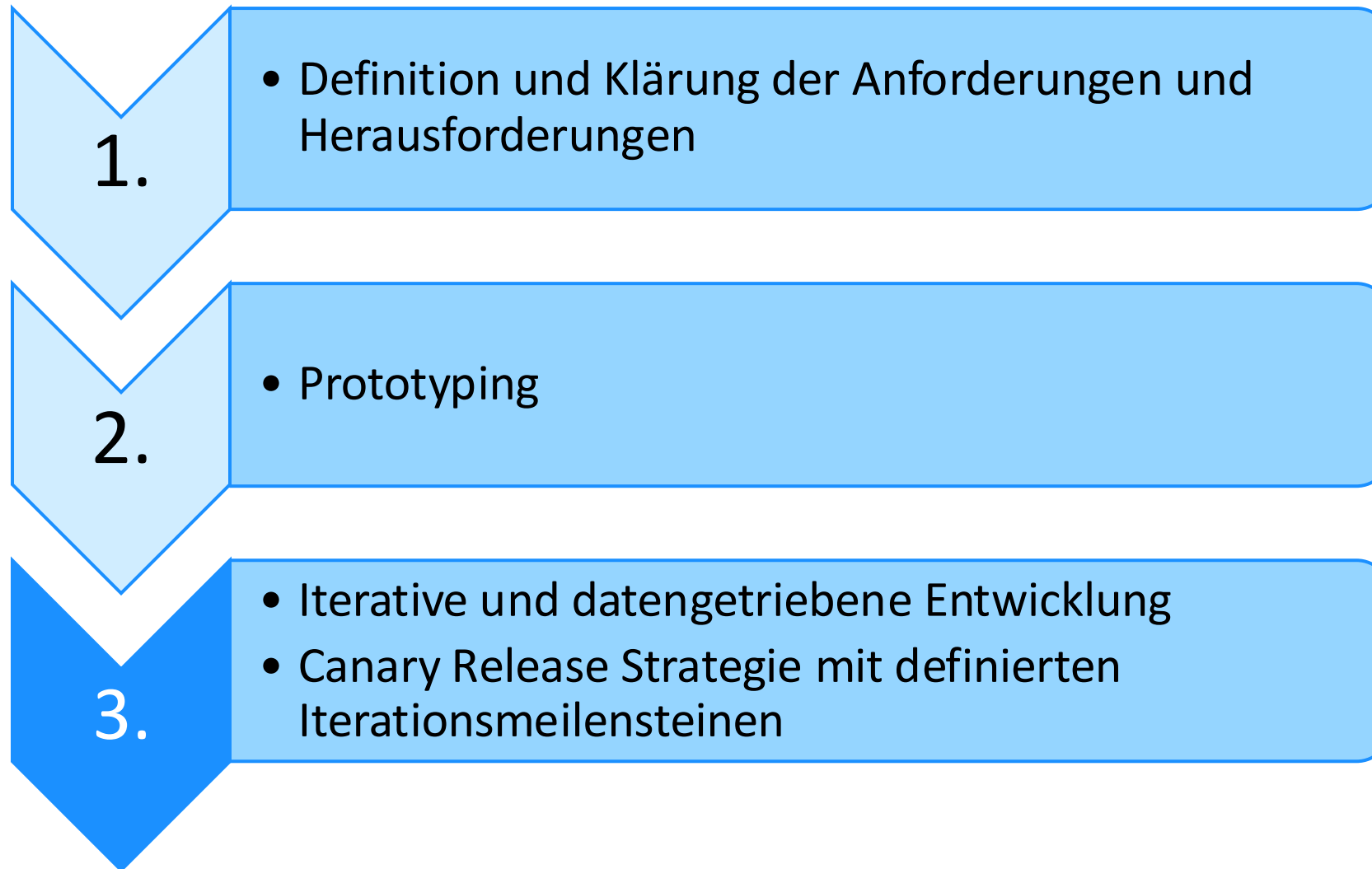
Wie sind wir vorgegangen



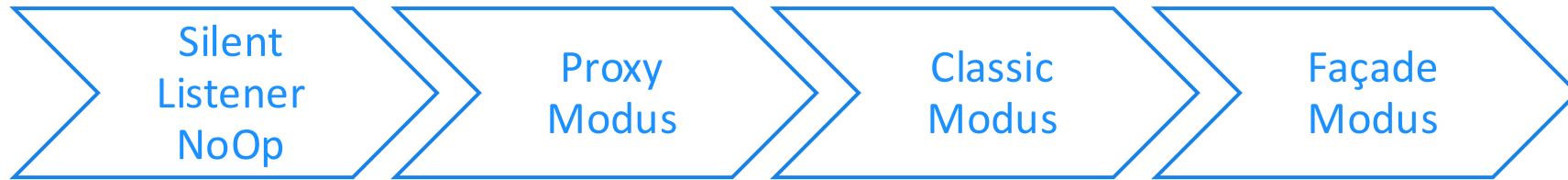
Feasibility Prototyping

- Verwendung von Hackathons für Feasibility Prototyping
 - Team/Cross-Team Hackathon im Büro ohne externe Meetings und wenig Prozessen (e.g. Scrum, Jira)
 - Jeder Hackathon Evaluation von mehreren Prototypen Ansätze
 - Tägliche Nachmittags Standups um neue Erkenntnisse zu teilen
 - Session mit Stakeholdern am Ende der Woche
 - Retrospektive und Zusammenfassung der Erfahrungen nach dem Hackathon mit nächsten Schritten

Wie sind wir vorgegangen

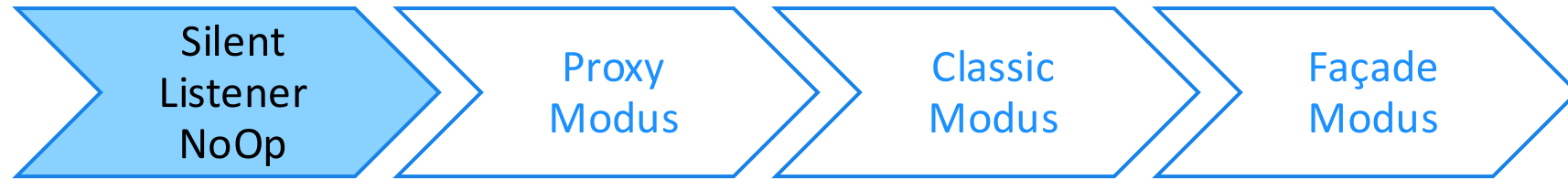


Iterative und datengetriebene Entwicklung



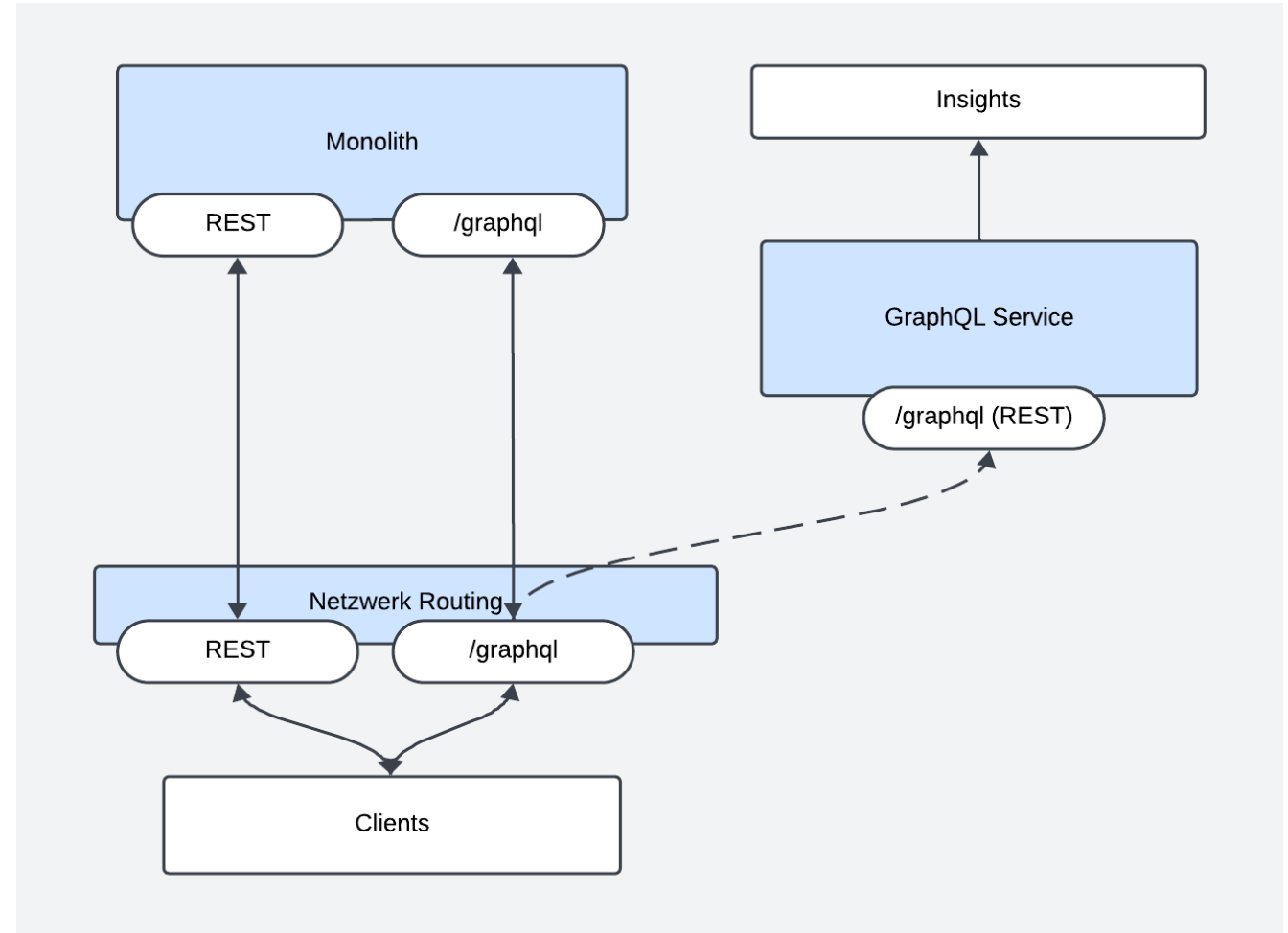
- Jeweils per Canary Release auf jeder Region einzeln ausgerollt von klein bis groß
 - Große Regionen nochmal unterteilt in Wellen
- Jeweils Pausen dazwischen für Kundenskripte oder Integrationen die nicht täglich laufen
- Konstantes Monitoring und Datenerhebung über Auswirkungen und Performance

Phase 0 - Meilenstein "Silent Listener NoOp"



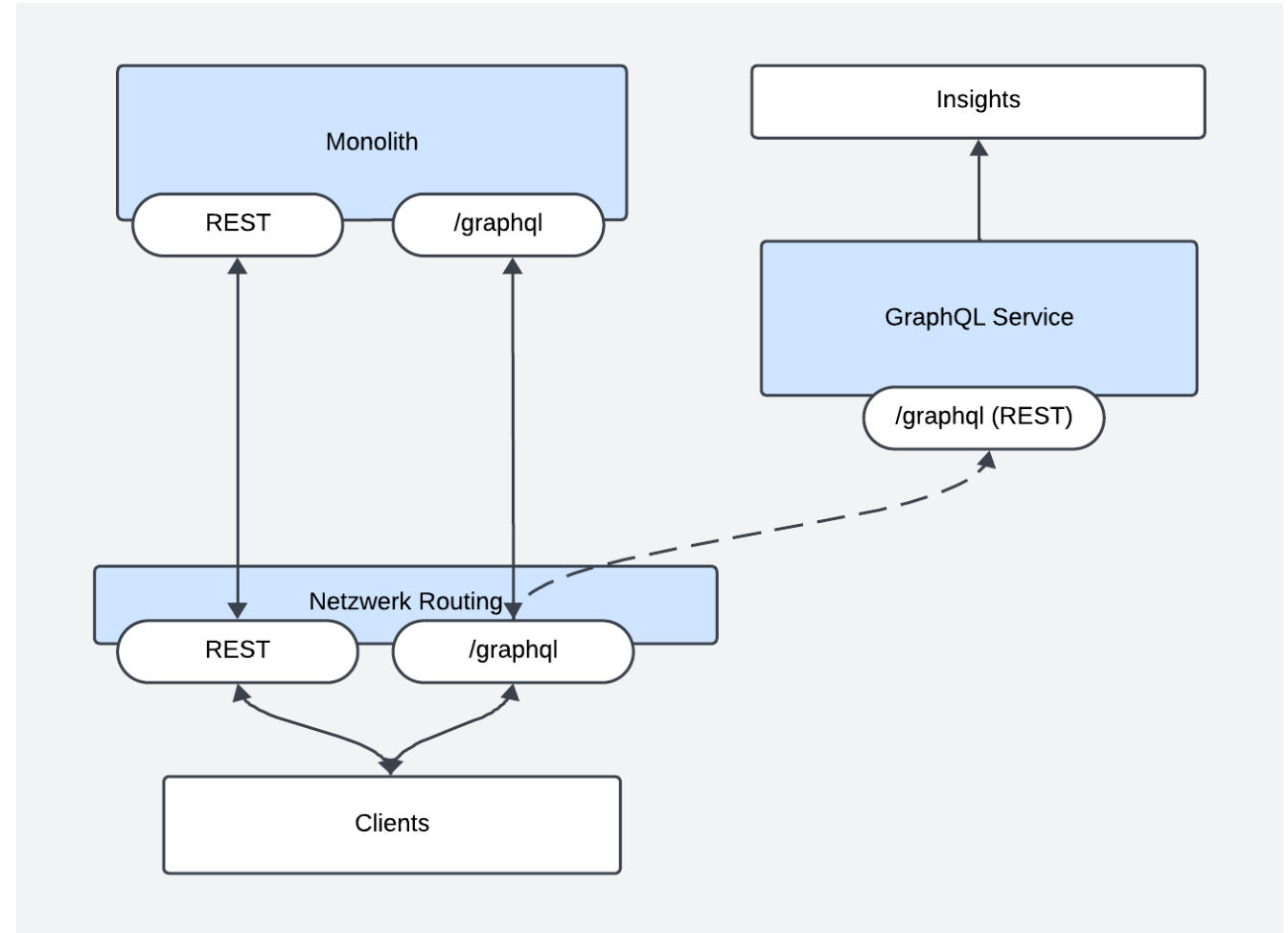
Phase 0 - Meilenstein "Silent Listener NoOp"

- „Mirroring“ im Netzwerkrouting
- Kein warten auf Antwort
- Analysengenerierung



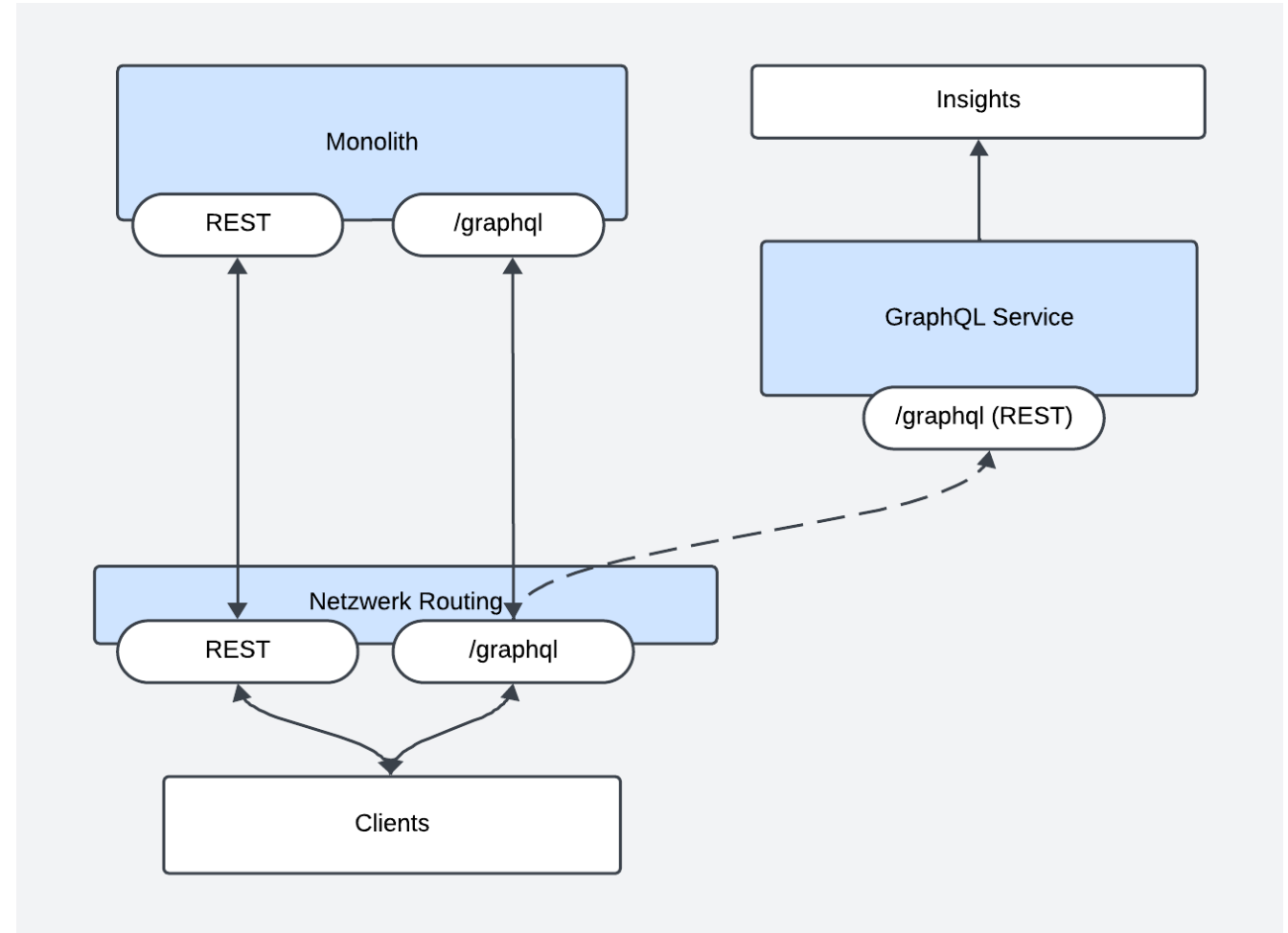
Phase 0 - Meilenstein "Silent Listener NoOp"

- Fragen die wir danach beantworten konnten:
 - Funktioniert neues Netzwerkrouting und wie sind die zusätzlichen Roundtripzeiten
 - Insights ermöglicht datengetriebene detaillierte Erkenntnisse runter bis auf Feldebene
 - Wieviele Pods werden für die reinen REST Verbindungen gebraucht

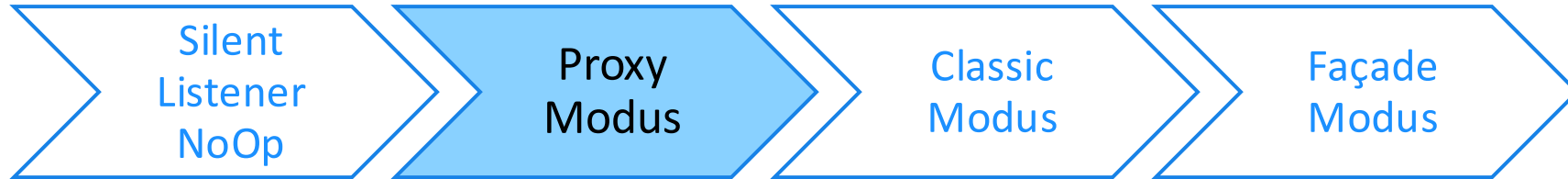


Phase 0 - Meilenstein "Silent Listener NoOp"

- Sachen die wir dabei gelernt haben:
 - Rewrite von Anfrage Routing
 - Mirroring ohne Warten auf Antworten
 - HTTP Server Optimierungen & Monitoring

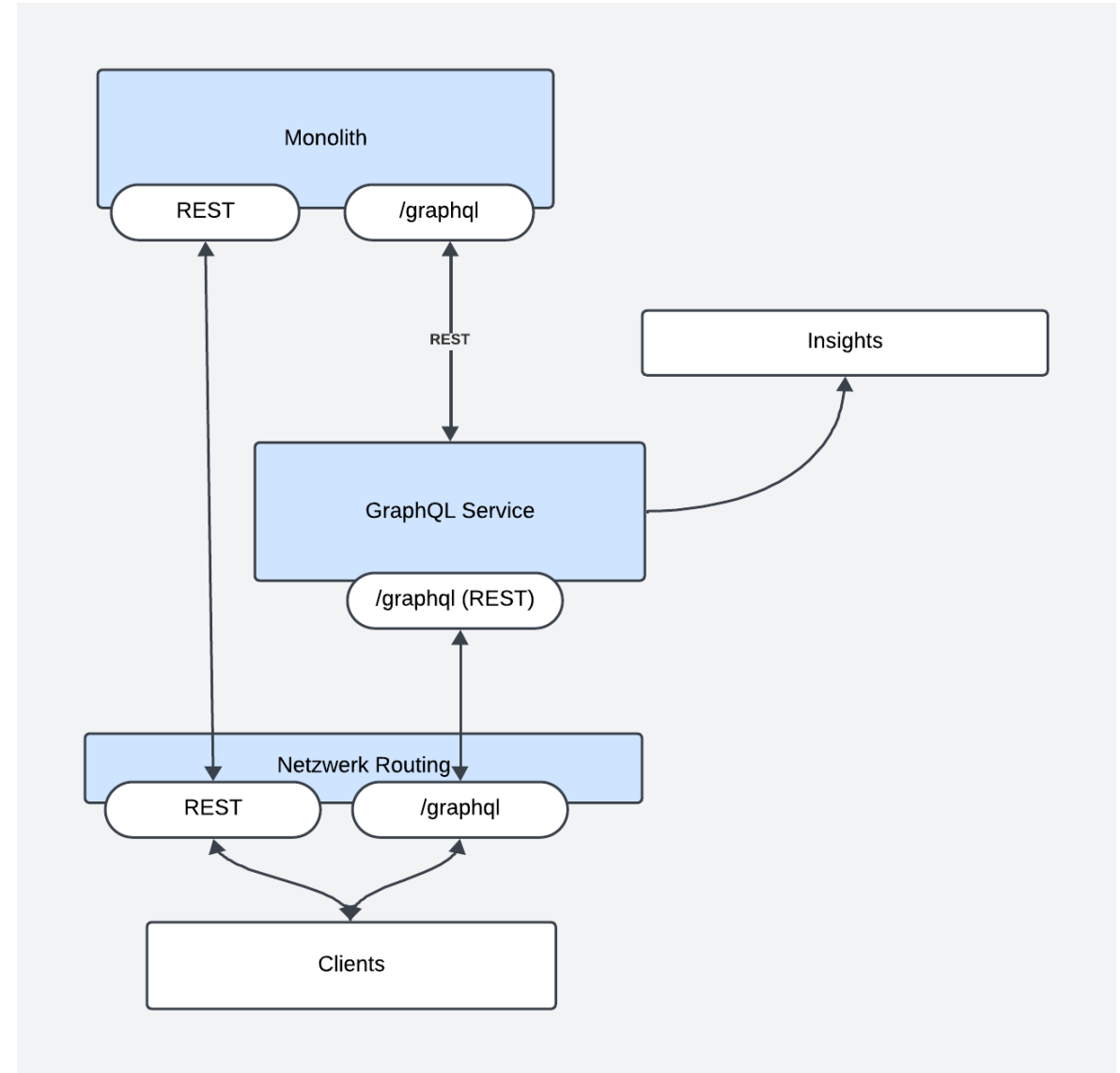


Phase 1 - Meilenstein "Proxy Modus"



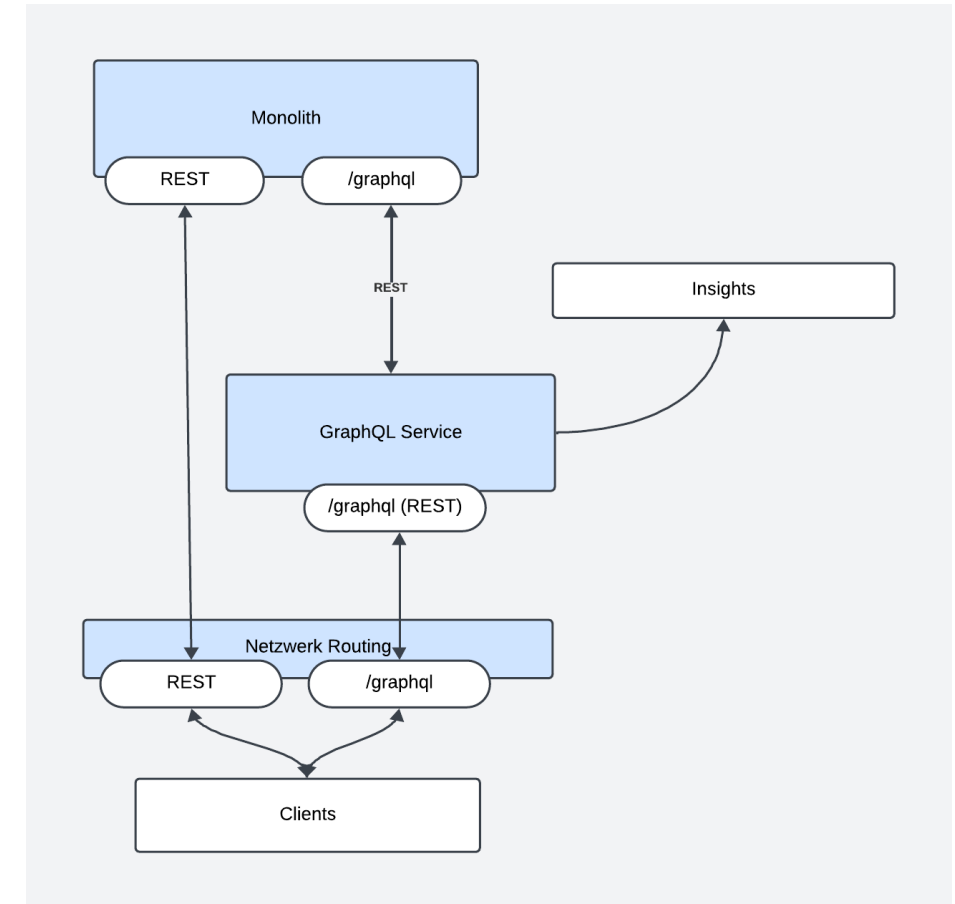
Phase 1 - Meilenstein "Proxy Modus"

- Service is verantwortlich für Anfragen
- Nachgelagert jede Anfrage gegen eigenes Schema validiert
- Änderung im Routing für alle Anfragen, ab dann ist der Service ein kritisches System



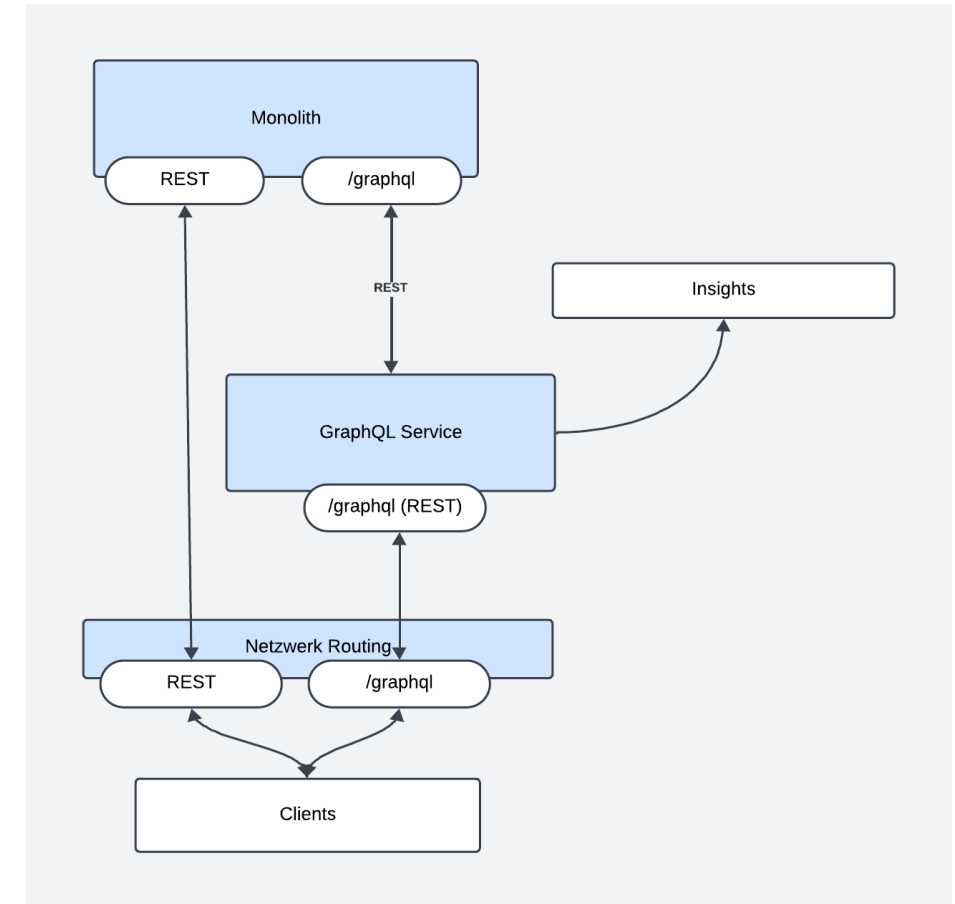
Phase 1 - Meilenstein "Proxy Modus"

- Fragen die wir danach beantworten konnten:
 - Wie sieht die Performance aus mit dem neuen Netzwerkrouting
 - Wieviele der Anfragen sind valide gegen aktuellen GraphQL Standard
 - Welche Bugs und Unterschiede finden wir in der GraphQL Implementierung
 - Wie müssen wir den neuen Service skalieren und schneiden
 - Aktualisierung Service Level Objectives

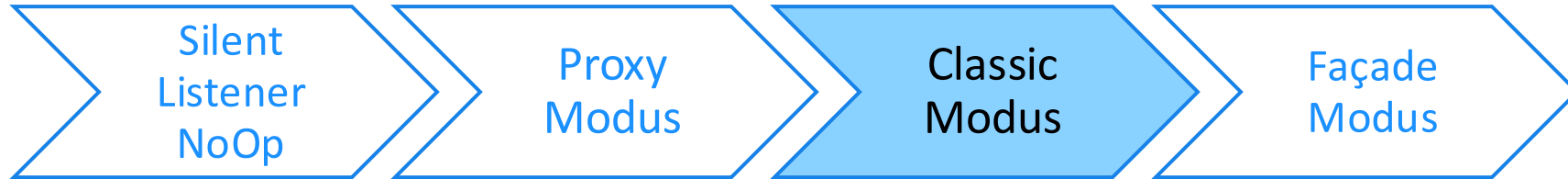


Phase 1 - Meilenstein "Proxy Modus"

- Sachen die wir dabei gelernt haben:
 - Weiterleiten von Headern
 - Skalierung und Monitoring von HTTP Requests & Request Handlern
 - Erste Einsichten in Anfragen die nicht GraphQL kompatibel sind
 - Schema-Caching und Optimierung des GraphQL Server Codes
 - Entwicklertooling für weitere Entwicklung, z.B. zusätzliche interne Header für Entwickler

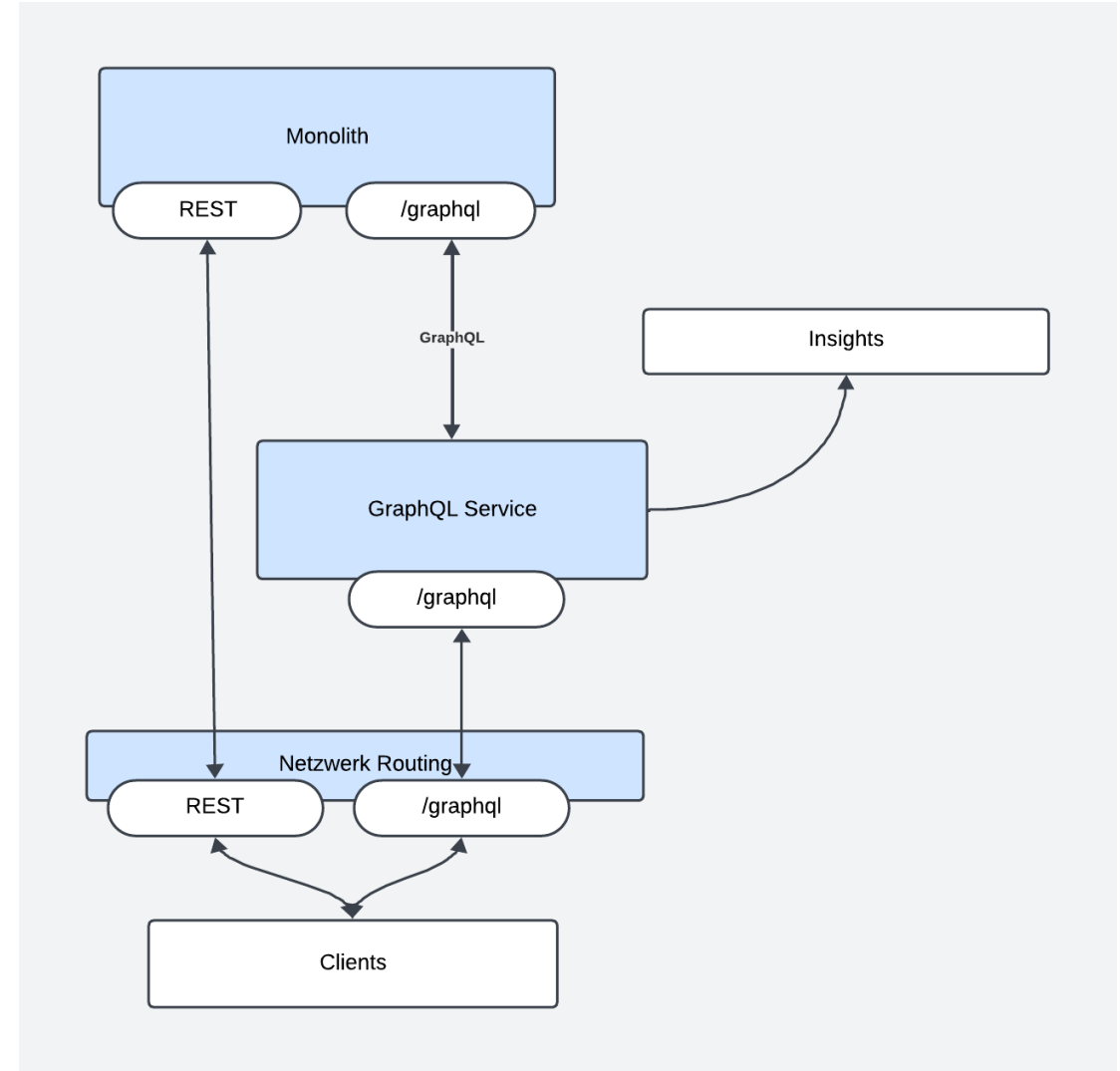


Phase 2 - Meilenstein "Classic Modus"



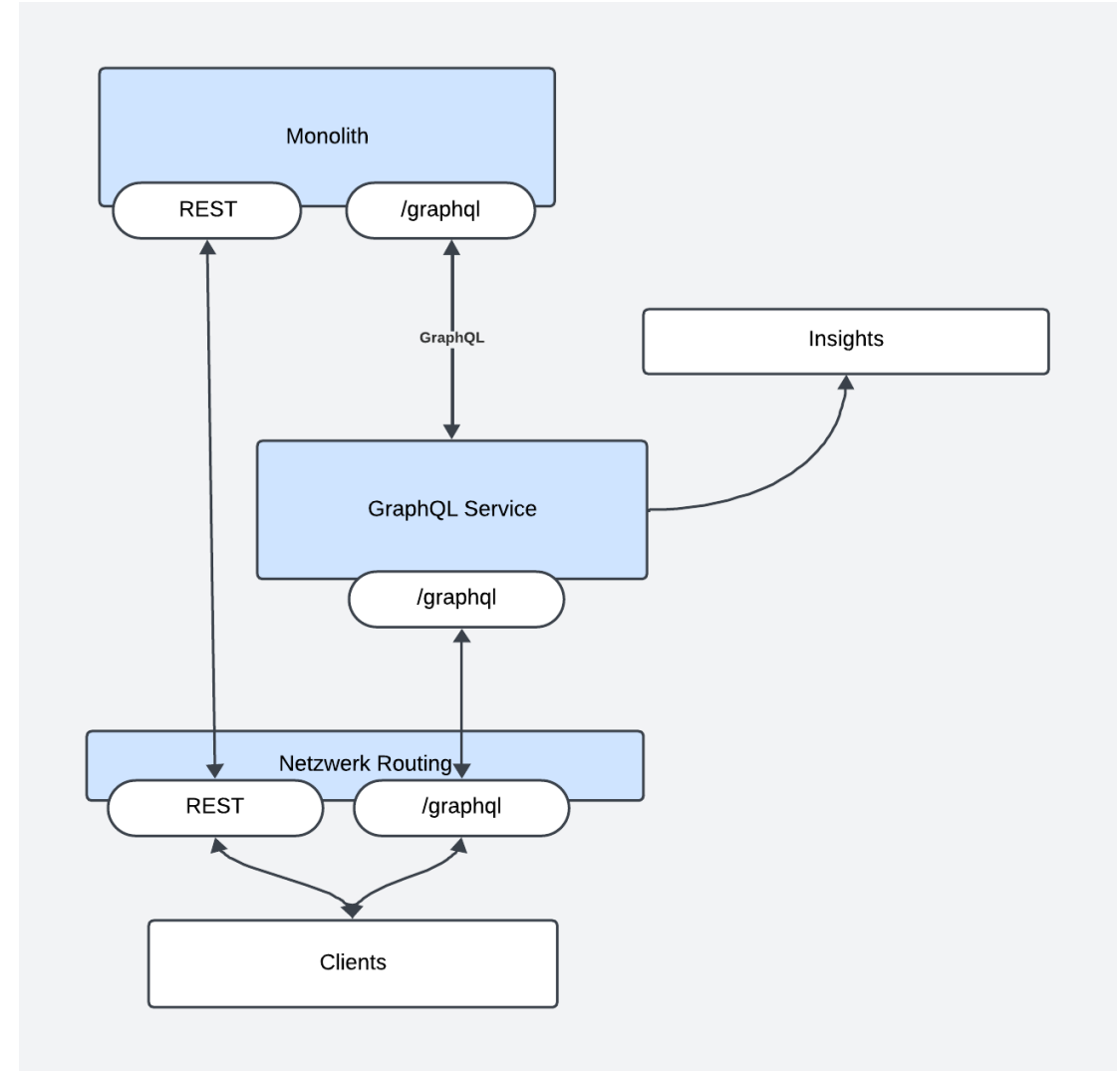
Phase 2 - Meilenstein "Classic Modus"

- Service ist Verantwortlich für das GraphQL Schema
- Bindet existierenden Service via GraphQL an
- Invalide Anfragen gehen wie im Proxy Modus via REST



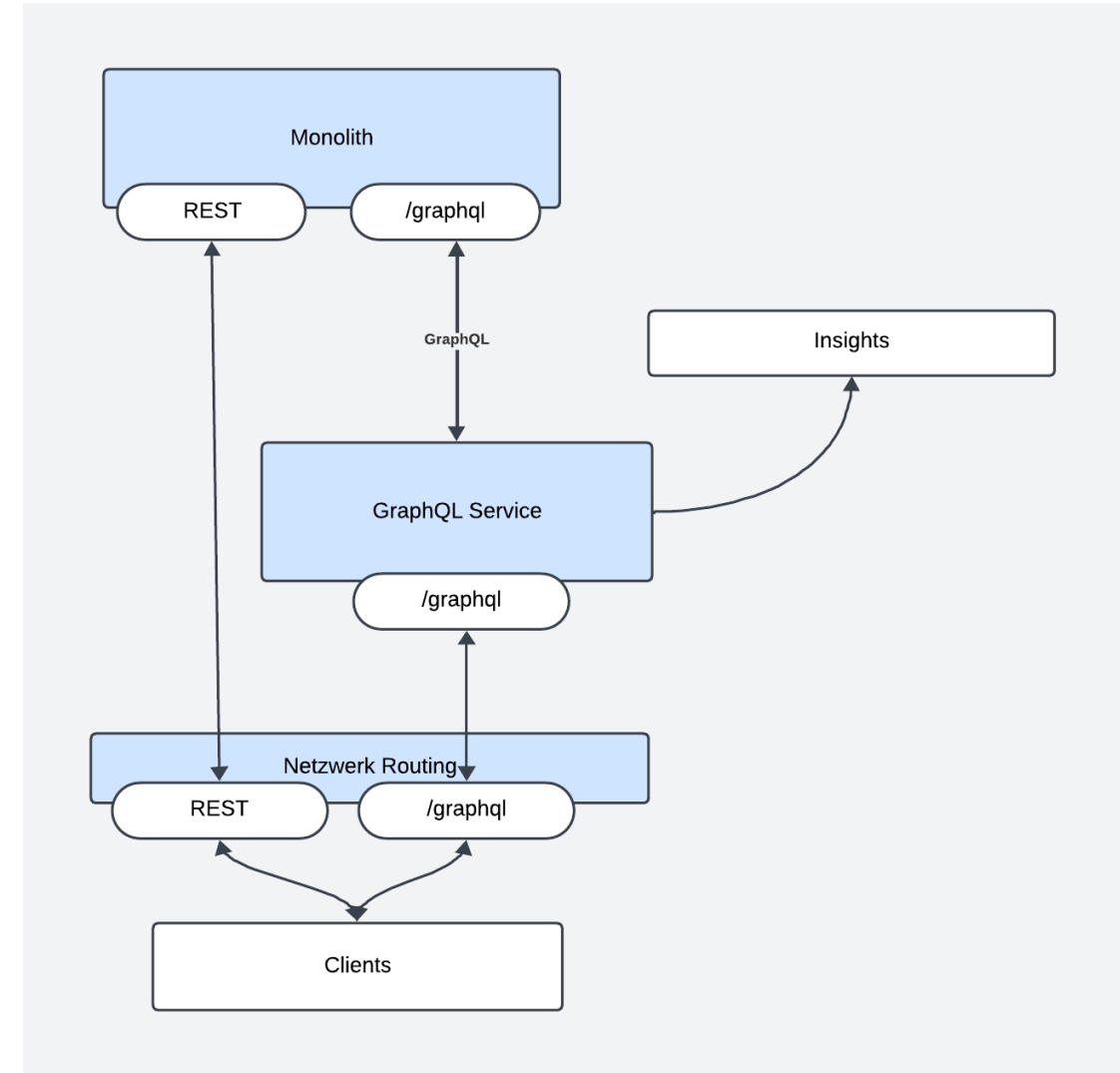
Phase 2 - Meilenstein "Classic Modus"

- Fragen die wir danach beantworten konnten:
 - Cross-Service Performance
 - Bottlenecks in der GraphQL Bibliothek
 - Standard-konformität der Anfragen



Phase 2 - Meilenstein "Classic Modus"

- Sachen die wir dabei gelernt haben:
 - Schema Caching und Delay zwischen Schemaänderungen
 - Skalierung der GraphQL Datafetcher
 - Bugs & Bottlenecks in der GraphQL Bibliothek
 - Cross-Service Performance Optimierungen
 - Iteration über Service Level Objectives
 - Benötigte Client Anpassungen wegen Security
 - Es gibt genug Requests die nicht GraphQL konform sind
 - ~2% initial
 - Kundenkommunikation mit Anpassungen notwendig

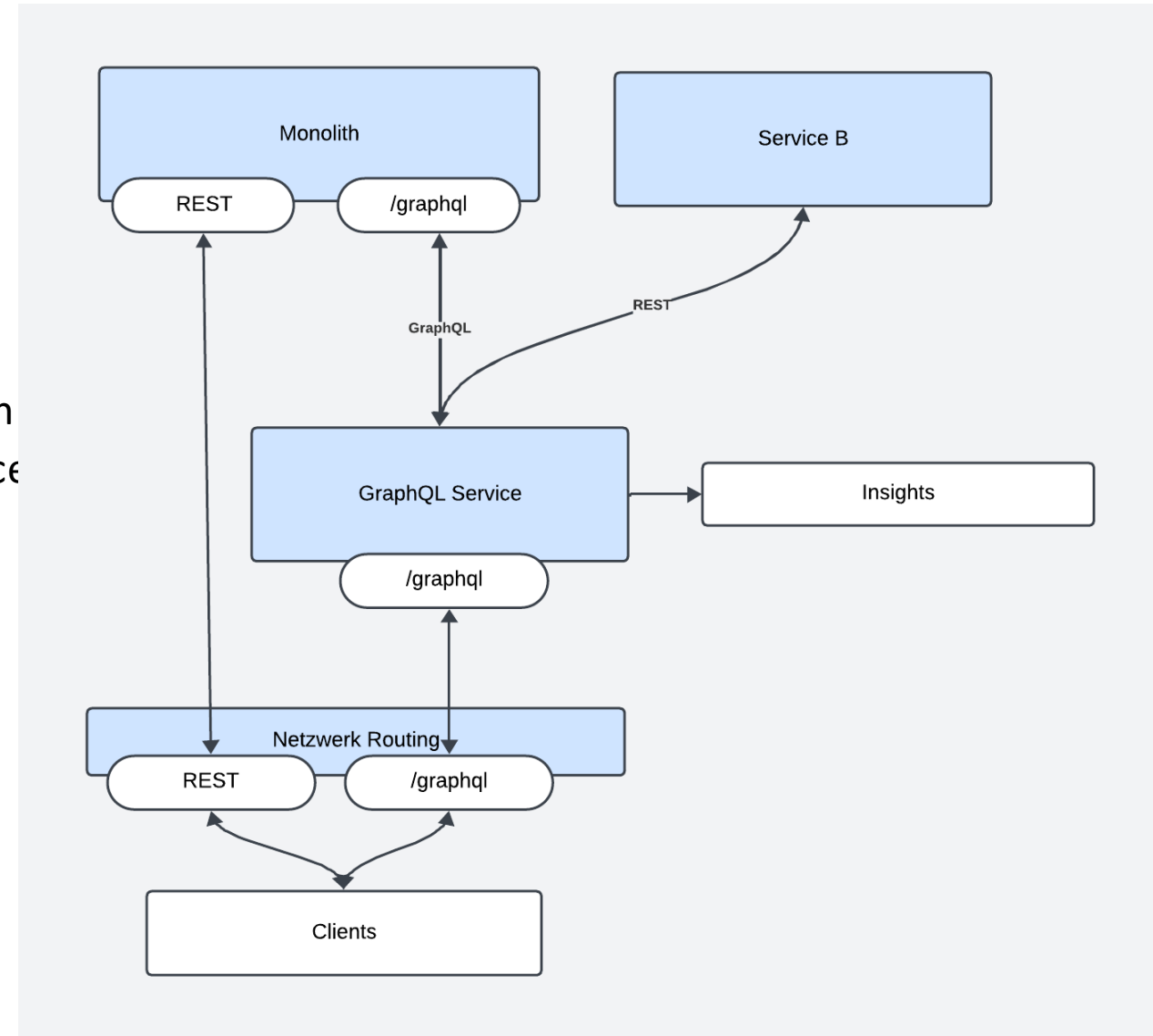


Phase 3 - Meilenstein "Facade Modus"

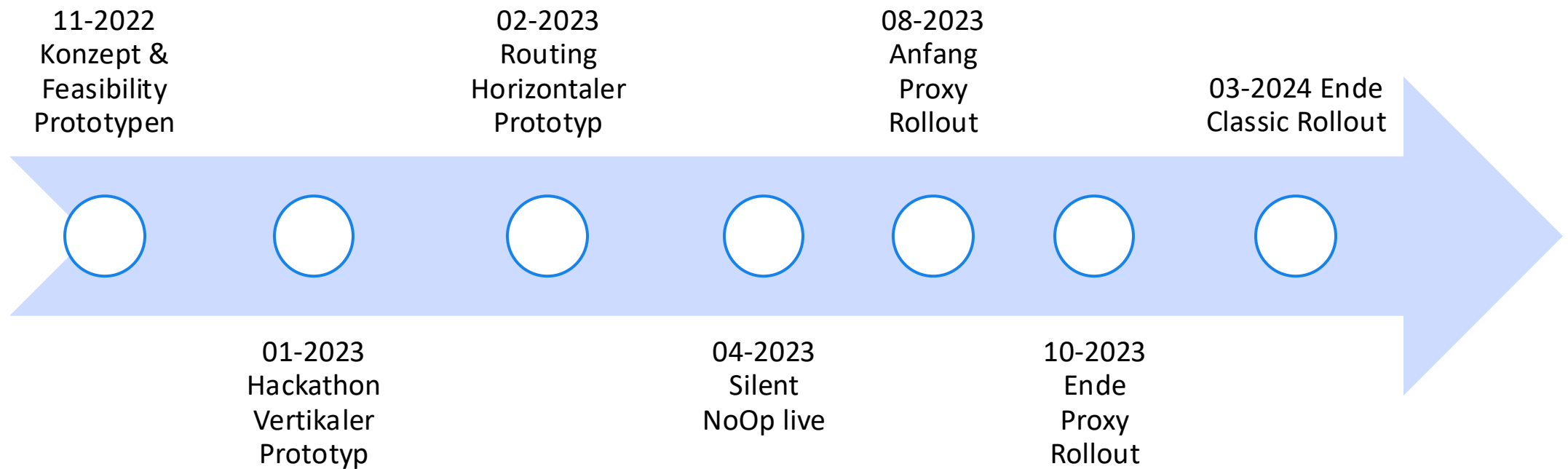


Phase 3 - Meilenstein "Facade Modus"

- Geplant aber noch nicht angefangen
- Geblockt von invaliden GraphQL Anfragen
- Ziel:
 - Ein valides dynamisches Schema pro Modells des Kunden
 - Anfragen per REST oder GraphQL an nachfolgende Service weitergibt



Wie lange hat es gedauert?



Zusammenfassung

- War die Migration nahtlos und ohne Kundenanpassungen?
 - Nahtlos ja. Ohne komplette Kundenanpassungen, nein, aber sinnvolle Anpassungen
 - 2% der täglichen Requests waren fehlerhaftes GraphQL (aktueller Stand ~0,35%)
- Frühe Erkenntnisse durch Prototyping & Daten haben essentiell geholfen in der Entwicklung
- Rollbacks beim Release waren erwartet worden und sind auch erfolgt
- Datengetriebenes Vorgehen war Gold wert
 - Fehler gefunden bevor sie bei Kunden aufgefallen sind
 - Statistiken über Validierung ermöglichen Zielgerichtet Unterstützung für Kunden
- Erweiterung der GraphQL-java library für Performancesteigerungen
- GraphQL-java auch in der aktuellen Version ist nicht immer Standard-konform

Fragen?

Contact information:

Jan Nonnen
jan.nonnen@leanix.net

