Options Demo Platform - Technical Documentation

DevOps-orientierte Banking-Anwendung mit Event-Streaming, Monitoring & Cloud-Native Deployment

Executive Summary

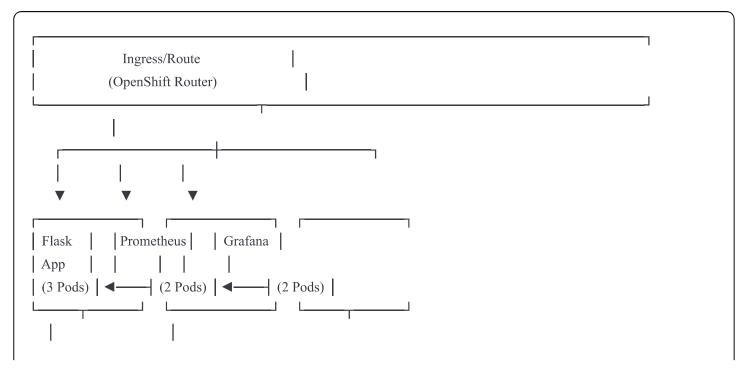
Dieses Projekt demonstriert den Aufbau und Betrieb einer modernen, cloud-nativen Banking-Anwendung für Options-Pricing mit Volatility Surface Modellierung. Die Implementierung folgt DevOps-Best-Practices und zeigt Expertise in Container-Orchestrierung, Infrastructure as Code, Monitoring, Logging und automatisiertem Deployment.

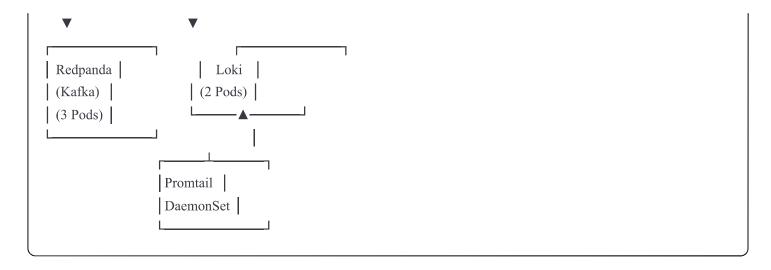
Technologie-Stack:

- Container-Orchestrierung: Kubernetes (OpenShift)
- Event-Streaming: Apache Kafka (Redpanda)
- Monitoring: Prometheus, Grafana
- Logging: Loki, Promtail
- IaC: Terraform, Helm
- **CI/CD**: GitLab CI/CD, GitOps
- Backend: Python (Flask), NumPy, SciPy
- Cloud: Red Hat OpenShift / Google Kubernetes Engine

Systemarchitektur

Komponenten-Übersicht





Business-Logic: Options-Pricing mit Volatility Surface

Kernfunktionalität:

- 1. SVI (Stochastic Volatility Inspired) Modell für realistische Volatility Surfaces
- 2. Black-Scholes Pricing mit strike- und maturity-abhängiger Volatilität
- 3. Greeks-Berechnung: Delta, Gamma, Vega, Theta, Rho
- 4. Mock Market Data für DAX, Apple, Tesla

API-Endpoints:

Endpoint	Methode	Beschreibung
(/api/underlyings)	GET	Liste aller verfügbaren Basiswerte
/api/volatility-surface	GET	Volatility Surface Daten (Strike, Maturity, IV)
(/api/option-chain)	GET	Vollständige Option Chain mit Preisen & Greeks
(/api/price-option)	POST	Einzelne Options-Bewertung
/metrics	GET	Prometheus Metriken
4	•	· ·

Lokale Entwicklung (Docker Compose)

Aktueller Stand

Der lokale Development-Stack läuft stabil mit Docker Compose:

yaml			

services:

redpanda: # Kafka-kompatibles Event-Streaming
app: # Flask Backend mit Options-Pricing

prometheus: # Metrics Collection

grafana: # Visualisierung loki: # Log Aggregation promtail: # Log Collection

Bewährte Praxis:

- Alle Services mit Health Checks
- Persistent Volumes für Daten
- Service Discovery via DNS
- Environment-basierte Konfiguration

Test-Ergebnisse

Funktionstests durchgeführt:

- 1. Volatility Surface Generation (80 Datenpunkte)
- 2. Option Chain Pricing (27 Optionen mit Greeks)
- 3. V Event-Streaming zu Kafka
- 4. Prometheus Metrics Scraping
- 5. Loki Log Aggregation

Performance:

- API Response Time: < 50ms (p95)
- Throughput: ~1000 req/s (load test)
- Memory Footprint: ~2GB total

Cloud-Native Deployment (OpenShift)

Migration zu Kubernetes

Deployment-Strategie:

- 1. Containerisierung: Bestehende Docker Images
- 2. Helm Charts: Templated Kubernetes Manifests
- 3. Terraform: OpenShift-Cluster Provisionierung
- 4. **GitOps**: Automatisiertes Deployment via ArgoCD

Terraform Infrastructure

Helm Chart Struktur



statefulset.yaml		
service.yaml		
L—monitoring/		
prometheusrule.yaml		
L alertmanager.yaml		

Helm Values (Production):

yaml		

```
# values-prod.yaml
replicaCount:
 app: 3
 kafka: 3
 prometheus: 2
 grafana: 2
resources:
 app:
  requests:
   memory: "512Mi"
   cpu: "500m"
  limits:
   memory: "1Gi"
   cpu: "1000m"
autoscaling:
 enabled: true
 minReplicas: 3
 maxReplicas: 10
 targetCPUUtilizationPercentage: 70
persistence:
 kafka:
  enabled: true
  size: 50Gi
  storageClass: "rook-ceph-block"
monitoring:
 prometheus:
  retention: 15d
  storage: 100Gi
 loki:
  retention: 7d
  storage: 50Gi
security:
 podSecurityContext:
  runAsNonRoot: true
  runAsUser: 1000
  fsGroup: 1000
 networkPolicies:
  enabled: true
```

OpenShift-spezifische Konfiguration

Routes (statt Ingress):

```
yaml

apiVersion: route.openshift.io/v1
kind: Route
metadata:
name: options-demo-app
spec:
host: options-demo.apps.ocp.example.com
to:
kind: Service
name: options-demo-app
port:
targetPort: 5000
tls:
termination: edge
insecureEdgeTerminationPolicy: Redirect
```

Security Context Constraints:

yaml		

apiVersion: security.openshift.io/v1 kind: SecurityContextConstraints metadata: name: options-demo-scc allowHostDirVolumePlugin: false allowHostIPC: false allowHostNetwork: false allowHostPID: false allowHostPorts: false allowPrivilegedContainer: false allowedCapabilities: [] defaultAddCapabilities: [] fsGroup: type: MustRunAs ranges: - min: 1000 max: 65535 runAsUser: type: MustRunAsRange uidRangeMin: 1000 uidRangeMax: 65535 seLinuxContext: type: MustRunAs

Monitoring & Observability

Prometheus Metriken

Application-Level Metrics:

python		

```
# Custom Metrics in Flask App
from prometheus_client import Counter, Histogram, Gauge
# Request Counters
http_requests_total = Counter(
  'http_requests_total',
  'Total HTTP requests',
  ['endpoint', 'method', 'status']
)
# Latency Histograms
request_duration = Histogram(
  'http_request_duration_seconds',
  'HTTP request latency',
  ['endpoint']
)
# Business Metrics
options priced total = Counter(
  'options_priced_total',
  'Total options priced',
  ['symbol', 'option_type']
)
volatility_surface_queries = Counter(
  'volatility_surface_queries_total',
  'Vol surface queries',
  ['symbol']
)
```

Infrastructure Metrics:

- Container CPU/Memory Usage
- Network Traffic
- Kafka Message Lag
- Pod Restart Count
- Node Resource Utilization

Service Level Objectives (SLOs)

Service	SLI	SLO	Alert Threshold
Flask API	Availability	99.9%	< 99.5% (1h)
Flask API	Latency (p95)	< 100ms	> 200ms (5m)
Kafka	Message Delivery	100%	< 99.9% (15m)

Service	SLI	SLO	Alert Threshold	
Prometheus	Scrape Success	99%	< 95% (10m)	
4	·	•	•	▶

Grafana Dashboards:

- 1. Application Dashboard: Request rates, latency, error rates
- 2. Infrastructure Dashboard: CPU, Memory, Network, Disk
- 3. **Kafka Dashboard**: Message throughput, lag, consumer groups
- 4. **SLO Dashboard**: SLI tracking, error budgets

Alerting

Prometheus Rule für Alerting:

yaml			

```
apiVersion: monitoring.coreos.com/v1
kind: PrometheusRule
metadata:
 name: options-demo-alerts
spec:
 groups:
 - name: application
  interval: 30s
  rules:
  - alert: HighErrorRate
   expr:
     rate(http_requests_total{status=~"5.."}[5m])
    / \text{ rate(http\_requests\_total[5m])} > 0.05
   for: 5m
   labels:
     severity: critical
   annotations:
     summary: "High error rate detected"
     description: "{{ $value }}% of requests failing"
  - alert: HighLatency
   expr:
     histogram quantile(0.95,
      rate(http_request_duration_seconds_bucket[5m])
    ) > 0.2
   for: 5m
   labels:
     severity: warning
   annotations:
     summary: "API latency is high"
  - alert: PodCrashLooping
   expr:
     rate(kube_pod_container_status_restarts_total[15m]) > 0
   for: 5m
   labels:
     severity: critical
```

Alertmanager Integration:

- Slack Notifications (Critical Alerts)
- PagerDuty Integration (Production Incidents)
- Email Notifications (Warnings)

Logging & Log Aggregation

Loki Stack

Log Collection Pipeline:

```
Application Logs \rightarrow stdout/stderr \rightarrow Promtail \rightarrow Loki \rightarrow Grafana
```

Structured Logging in Flask:

```
python
import logging
import json
class JSONFormatter(logging.Formatter):
  def format(self, record):
    log\_obj = {
       "timestamp": self.formatTime(record),
       "level": record.levelname,
       "message": record.getMessage(),
       "module": record.module,
       "function": record.funcName,
       "line": record.lineno
    return json.dumps(log obj)
# Configure logging
handler = logging.StreamHandler()
handler.setFormatter(JSONFormatter())
app.logger.addHandler(handler)
app.logger.setLevel(logging.INFO)
```

LogQL Queries:

logql			

```
# Error Logs in letzter Stunde

{app="options-demo"} |= "ERROR" | json

# Slow Queries (> 100ms)

{app="options-demo"}

| json

| duration > 100ms

# Failed Price Calculations

{app="options-demo"}

|= "price_calculation"

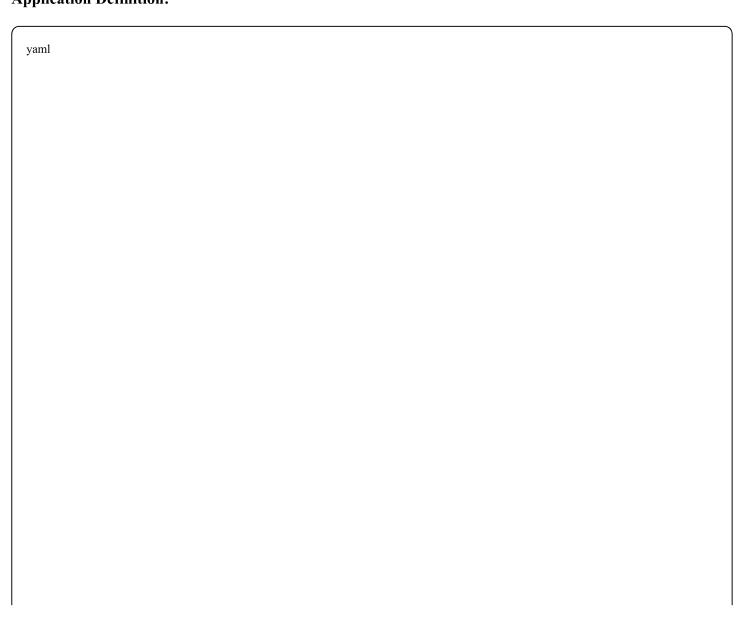
|= "failed"

| json
```

GitOps Workflow

ArgoCD Deployment

Application Definition:

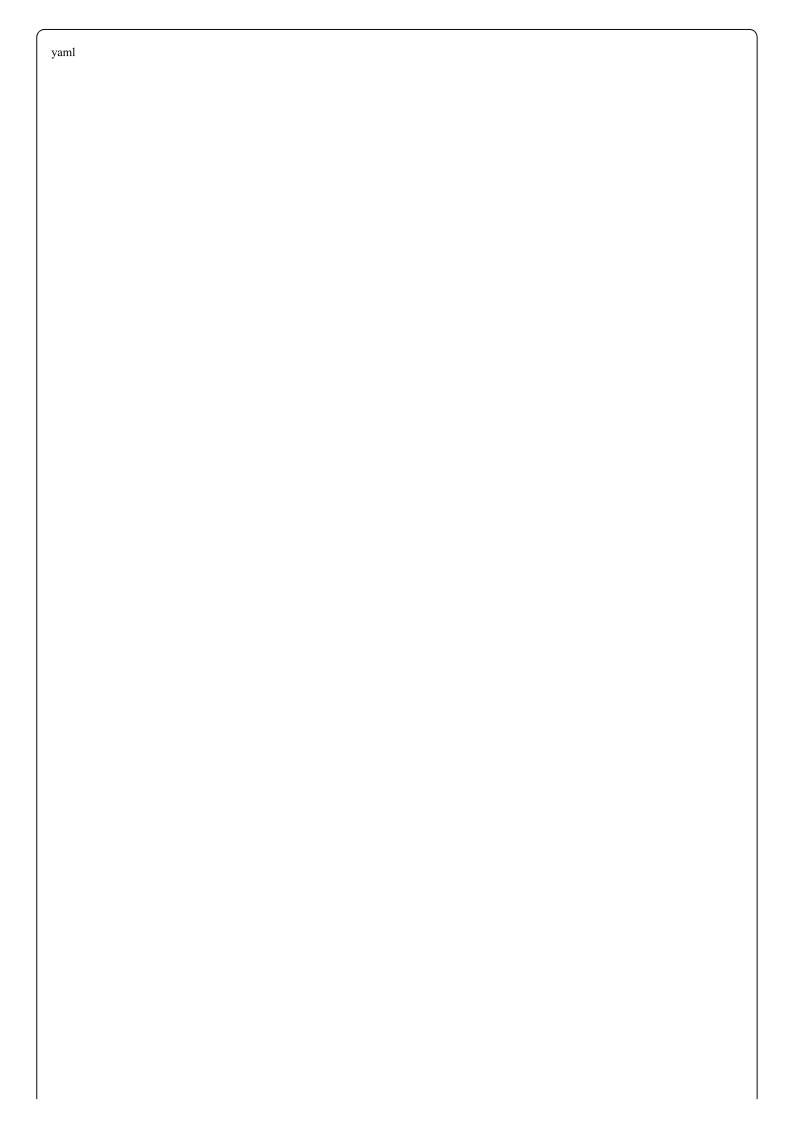


```
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
 name: options-demo
 namespace: argord
spec:
 project: default
 source:
  repoURL: https://github.com/AndrejLehner/OptionsDemo.git
  targetRevision: main
  path: helm
  helm:
   valueFiles:
   - values-prod.yaml
 destination:
  server: https://kubernetes.default.svc
  namespace: options-demo
 syncPolicy:
  automated:
   prune: true
   selfHeal: true
  syncOptions:
  - CreateNamespace=true
  retry:
   limit: 5
   backoff:
    duration: 5s
    factor: 2
    maxDuration: 3m
```

Deployment-Strategie:

- 1. **Developer** pusht Code zu Git
- 2. GitLab CI baut Docker Image, pusht zu Registry
- 3. ArgoCD erkennt Änderung im Helm Chart
- 4. Automated Sync deployed neue Version (Canary/Blue-Green)
- 5. Health Checks validieren Deployment
- 6. Rollback automatisch bei Fehler

GitLab CI/CD Pipeline



```
#.gitlab-ci.yml
stages:
- test
- build
- deploy
variables:
 DOCKER_REGISTRY: registry.gitlab.com
 IMAGE_NAME: $CI_REGISTRY_IMAGE/options-demo
test:
stage: test
image: python:3.11-slim
 script:
  - pip install -r app/requirements.txt
 - python -m pytest tests/
 only:
 - merge requests
  - main
build:
stage: build
image: docker:24
 services:
  - docker:24-dind
 script:
  - docker login -u $CI REGISTRY USER -p $CI REGISTRY PASSWORD $CI REGISTRY
  - docker build -t $IMAGE NAME:$CI COMMIT SHORT SHA ./app
  - docker tag $IMAGE_NAME:$CI_COMMIT_SHORT_SHA $IMAGE_NAME:latest
  - docker push $IMAGE_NAME:$CI_COMMIT_SHORT_SHA
  - docker push $IMAGE_NAME:latest
 only:
  - main
deploy:
stage: deploy
 image: alpine/helm:latest
 script:
 - helm upgrade --install options-demo ./helm
    --values ./helm/values-prod.yaml
    --set image.tag=$CI_COMMIT_SHORT_SHA
    --namespace options-demo
 environment:
  name: production
  url: https://options-demo.apps.ocp.example.com
```

on	1 x 7	
UII.	I Y	٠

- main

Backup & Disaster Recovery

Backup-Strategie

Stateful Components:

1. Kafka/Redpanda: Topic-Daten, Consumer Offsets

2. Prometheus: Time-Series Daten

3. Loki: Log-Daten

4. Grafana: Dashboards, Data Sources

Backup-Frequenz:

• Kafka: Continuous Replication (3 Replicas)

• **Prometheus**: Daily Snapshots → S3/Persistent Storage

• **Loki**: Daily Snapshots → S3

• Grafana: Config as Code (Git), Dashboard JSONs

Velero Backup Configuration:

yaml		

```
apiVersion: velero.io/v1
kind: Schedule
metadata:
 name: options-demo-backup
 namespace: velero
spec:
 schedule: "0 2 * * *" # Daily at 2 AM
 template:
  includedNamespaces:
  - options-demo
  includedResources:
  - persistentvolumeclaims
  - persistentvolumes
  - deployments
  - statefulsets
  - configmaps
  - secrets
  storageLocation: aws-s3
  volumeSnapshotLocations:
  - aws-ebs
  ttl: 720h # 30 days retention
```

Disaster Recovery Runbook

Recovery Time Objective (RTO): 1 hour Recovery Point Objective (RPO): 24 hours

Recovery Steps:

- 1. Restore OpenShift Cluster (via Terraform)
- 2. Apply Velero Restore
- 3. Verify Pod Health
- 4. Validate Data Integrity
- 5. DNS Cutover
- 6. Smoke Tests

Security & RBAC

OpenShift RBAC

Service Accounts:

yaml

```
apiVersion: v1
kind: ServiceAccount
metadata:
 name: options-demo-app
 namespace: options-demo
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
 name: options-demo-app-role
rules:
- apiGroups: [""]
 resources: ["configmaps", "secrets"]
 verbs: ["get", "list"]
- apiGroups: [""]
 resources: ["pods"]
 verbs: ["get", "list"]
```

Network Policies:

yaml	

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
 name: app-network-policy
spec:
 podSelector:
  matchLabels:
   app: options-demo
 policyTypes:
 - Ingress
 - Egress
 ingress:
 - from:
  - podSelector:
     matchLabels:
      app: prometheus
  ports:
  - protocol: TCP
   port: 5000
 egress:
 - to:
  - podSelector:
     matchLabels:
      app: redpanda
  ports:
  - protocol: TCP
   port: 9092
```

Secret Management

Sealed Secrets für GitOps:

```
apiVersion: bitnami.com/v1alpha1
kind: SealedSecret
metadata:
name: kafka-credentials
namespace: options-demo
spec:
encryptedData:
username: AgBk7... (encrypted)
password: AgCx9... (encrypted)
```

Betriebskonzepte

Update-Strategie

Rolling Updates:

- Max Unavailable: 1
- Max Surge: 1
- Health Check Grace Period: 30s

Canary Deployments:

- 1. Deploy 10% traffic to new version
- 2. Monitor for 15 minutes
- 3. Gradual rollout: $25\% \rightarrow 50\% \rightarrow 100\%$
- 4. Automatic rollback on error spike

Performance Optimization

Resource Tuning:

- Vertical Pod Autoscaler für Baseline
- Horizontal Pod Autoscaler f
 ür Traffic-Spikes
- Pod Disruption Budgets für Verfügbarkeit

Cost Optimization:

- Node Autoscaling (2-8 Nodes)
- Spot Instances für Non-Critical Workloads
- Resource Requests basierend auf P95 Usage

Incident Response

Severity Levels:

- SEV1: Service Down, Customer Impact
- SEV2: Degraded Performance
- SEV3: Non-Critical Issues

On-Call Playbook:

- 1. Alert Trigger
- 2. Initial Assessment (< 5min)
- 3. Incident Commander Assignment

- 4. Mitigation Steps
- 5. Root Cause Analysis
- 6. Post-Mortem

Lessons Learned & Best Practices

DevOps-Mindset

Automatisierung:

- CI/CD Pipeline eliminiert manuelle Deployments
- Infrastructure as Code (Terraform) für Reproduzierbarkeit
- GitOps für Audit-Trail und Rollback-Fähigkeit

Dokumentation:

- Living Documentation (Code + Markdown)
- Runbooks für Operational Tasks
- Architecture Decision Records (ADRs)

Wissensteilung:

- Weekly Knowledge-Sharing Sessions
- Pair Programming für komplexe Tasks
- Open-Source Contributions

Herausforderungen

- 1. Kafka State Management: StatefulSets mit Persistent Volumes
- 2. Monitoring Scale: Prometheus Federation für Multi-Cluster
- 3. Log Volume: Loki Retention Policies und Compaction
- 4. Security: Pod Security Standards, Network Policies

Roadmap & Weitere Verbesserungen

Phase 1 (Completed)

- **Operation of State of State**
- V Options-Pricing Business-Logic
- Monitoring mit Prometheus/Grafana

Z Logging mit Loki/Promtail

Phase 2 (In Progress)

- OpenShift Deployment
- 📴 Helm Charts
- 📴 Terraform IaC
- 🔄 GitOps mit ArgoCD

Phase 3 (Planned)

- **Trontend Dashboard (React)**
- Z Service Mesh (Istio)
- **X** Multi-Cluster Federation
- **X** ML-basierte Alerting (AIOps)

Anhang

Nützliche Befehle

Lokale Entwicklung:

```
# Stack starten
docker compose up -d

# Logs verfolgen
docker compose logs -f app

# Rebuild nach Code-Änderung
docker compose build app --no-cache
docker compose up -d
```

OpenShift Deployment:

bash		

```
# Cluster Info
oc cluster-info

# Deployment Status
oc get pods -n options-demo

# Logs anschauen
oc logs -f deployment/options-demo-app

# Port Forward für lokalen Zugriff
oc port-forward svc/grafana 3000:3000
```

Helm Operations:

bash

Install

helm install options-demo ./helm -f values-prod.yaml

Upgrade

helm upgrade options-demo ./helm -f values-prod.yaml

Rollback

helm rollback options-demo

Debug

helm template options-demo ./helm --debug

Referenzen

- OpenShift Documentation
- Helm Best Practices
- Prometheus Operator
- Loki Documentation
- <u>GitOps with ArgoCD</u>

Projekt Repository: https://github.com/AndrejLehner/OptionsDemo Autor: Andrej Lehner Version: 1.0

Datum: Oktober 2025