Verteilte Systeme und Cloud Technologien - Laborübung 01: Grundlagen und Architektur

Wintersemester 2025
(c) 2025
Simon Kranzer, Gerald Lochner

1. Einführung

Diese erste Laborübung dient der Einarbeitung in die Domäne der Bürgerenergiegemeinschaften und der Konzeption einer verteilten Systemarchitektur. Das zu entwickelnde System soll die Verwaltung, Abrechnung und Analyse von Energiegemeinschaften ermöglichen.

2. Ziele der Übung

- Verständnis für die Anforderungen von Bürgerenergiegemeinschaften entwickeln
- Eine Microservices-basierte Softwarearchitektur entwerfen
- Lokale Entwicklungsumgebung mit Container-Orchestrierung aufsetzen
- Grundlagen für die weiteren Laborübungen schaffen

3. Technologische Rahmenbedingungen

Folgende Technologien sind für das Gesamtprojekt vorgegeben:

- **Programmiersprache**: C# mit .NET 9
- **Message Broker**: RabbitMQ für asynchrone Kommunikation
- **Datenbank**: MongoDB für Service-spezifische Datenhaltung
- **Container-Orchestrierung**: k3s (Lightweight Kubernetes)
- **Containerisierung**: Docker

Weitere Technologieentscheidungen (z.B. Authentifizierung, API Gateway) werden im Rahmen der Architekturkonzeption getroffen.

4. Aufgabenstellung

4.1. Aufgabe 1: Domänenanalyse Energiegemeinschaften (30%)

Erarbeiten Sie ein fundiertes Verständnis für Bürgerenergiegemeinschaften in Österreich:

4.1.1. Recherche

- Studieren Sie das Erneuerbaren-Ausbau-Gesetz (EAG) in Bezug auf Energiegemeinschaften
- Analysieren Sie die Rollen in einer Energiegemeinschaft (Erzeuger, Verbraucher, Prosumer)
- Verstehen Sie die Abrechnungsmodelle und regulatorischen Anforderungen

4.1.2. Anforderungsanalyse

Identifizieren und dokumentieren Sie:

- **Funktionale Anforderungen**:
  - Datenerfassung von Smart Metern
  - Abrechnungslogik innerhalb der Gemeinschaft
  - Verwaltung von Teilnehmern
  - Energiebilanzierung
  - Reporting und Visualisierung
- **Nicht-funktionale Anforderungen**:
  - Skalierbarkeit (Anzahl Teilnehmer, Datenpunkte)
  - Verfügbarkeit und Ausfallsicherheit
  - Datenschutz und Datensicherheit
  - Performance (Latenz, Durchsatz)
  - Interoperabilität mit bestehenden Systemen

4.1.3. Deliverable

Erstellen Sie einen Abschnitt "Domäne und Anforderungen" (2-3 Seiten) mit:

- Übersicht über Energiegemeinschaften
- Use Case Diagramm
- Priorisierte Anforderungsliste

4.2. Aufgabe 2: Softwarearchitektur Teil 1 für verteiltes System (50%)

Entwerfen Sie eine Microservices-Architektur für das Energiegemeinschafts-System.

4.2.1. Service-Identifikation

Definieren Sie die benötigten Microservices (mind. 4) basierend auf Domain-Driven Design Prinzipien.

Mögliche Services: * **Community Management Service**: Verwaltung von Teilnehmern und Rollen * **Billing Service**: Abrechnung und Fakturierung

4.2.2. Architekturdiagramme

Erstellen Sie folgende Diagramme:

- **System Context Diagram** (z. B. anhand C4 Level 1)
- **Sequenzdiagramm** für einen Hauptprozess (z.B. Energieabrechnung)

4.2.3. Kommunikationsdesign

Spezifizieren Sie die Service-Kommunikation:

- **Synchrone Kommunikation**: REST APIs oder gRPC
- **Asynchrone Kommunikation**: Event-basiert über RabbitMQ
  - Exchange Types (Direct, Topic, Fanout)
  - Queue Design
  - Message Schemas

4.2.4. Deliverable

Erstellen Sie einen Abschnitt "Softwarearchitektur" (4-5 Seiten) mit:

- Service-Beschreibungen
- Architekturdiagrammen
- Kommunikationskonzept
- Technologie-Entscheidungstabelle mit Begründungen

4.3. Aufgabe 3: Entwicklungsumgebung Setup (20%)

Richten Sie die lokale Entwicklungsumgebung ein. Wir werden diese Umgebung in den nächsten Laborübungen erweitern und nutzen.

4.3.1. Voraussetzungen

Installieren Sie folgende Tools, bevor Sie beginnen:

- **Docker** (aktuelle Version erforderlich, mind. 24.x)
  - macOS: Docker Desktop für Mac

    - Windows: Docker Desktop für Windows
    - Linux: Docker Engine für Linux
- **kubectl**: Kubernetes Kommandozeilen-Tool
- **helm**: Helm Package Manager
- **MongoDB Compass** (optional, für DB-Visualisierung): MongoDB Compass
- **.NET 9 SDK**: .NET 9 SDK
- **Visual Studio Code** (empfohlen): VS Code
- **JetBrains Rider** (IDE für .NET): JetBrains Rider
- **Git**: Git
- **Postman** (optional, für API Tests): Postman
- **Lens** (optional, Kubernetes GUI): Lens IDE

Sie erhalten von JetBrains eine kostenlose Lizenz für Rider als Student.

Windows-Nutzer arbeiten in WSL2/Ubuntu: Nach Docker Desktop Installation führen Sie alle weiteren Schritte in WSL2 (Ubuntu) aus. Docker Desktop stellt Docker automatisch in WSL2 zur Verfügung: * Nach Installation: Docker Desktop Settings → Resources → WSL Integration → Ubuntu aktivieren

Falls Sie die WSL Integration nachträglich aktivieren, müssen Sie WSL neu starten:

```
# In Windows PowerShell als Administrator:
wsl --shutdown
# Dann WSL/Ubuntu wieder starten:
wsl
```

4.3.2. k3s Installation mit k3d

k3d ist ein Tool zum einfachen Erstellen und Verwalten von k3s Clustern in Docker-Containern. Es eignet sich hervorragend für lokale Entwicklungsumgebungen. k3s ist eine leichtgewichtige Kubernetes-Distribution, die speziell für IoT- und Edge-Computing entwickelt wurde, aber auch ideal für lokale Entwicklungsumgebungen ist.

Alternativ können Sie auch Minikube oder Kind verwenden, k3d ist jedoch besonders einfach zu handhaben.

Es gibt einen Cheatsheet für kubectl: kubectl Spickzettel

```
# Installation k3d (https://k3d.io)
# macOS
brew install k3d

# Linux
```

```
curl -s https://raw.githubusercontent.com/k3d-io/k3d/main/install.sh | bash

# Windows (WSL2)
curl -s https://raw.githubusercontent.com/k3d-io/k3d/main/install.sh | bash
```

4.3.3. Cluster erstellen

```
# Erstellen eines k3s Clusters mit 3 Worker Nodes
k3d cluster create energy-community \
  --servers 1 \
  --agents 3 \
  --port "8080:80@loadbalancer" \
  --port "8443:443@loadbalancer" \
  --volume "$(pwd)/data:/data@all" \
  --registry-create energy-registry:5000

# Verifizieren
kubectl get nodes
kubectl get namespaces
```

Es wird in diesem Script im Arbeitsverzeichnis ein Ordner "data" (siehe `$(pwd)/data`) erstellt, der die Daten der Services in Kubernetes persistiert. Diesen Ordner können Sie später löschen, wenn Sie den Cluster entfernen.

Cluster löschen

Wenn Sie den Cluster nicht mehr benötigen, können Sie ihn mit folgendem Befehl löschen:

```
k3d cluster delete energy-community
```

4.3.4. Basis-Services deployen

```
# Namespace erstellen
kubectl create namespace energy-system

# RabbitMQ installieren (Helm)
helm repo add bitnami https://charts.bitnami.com/bitnami
helm install rabbitmq bitnami/rabbitmq \
  --namespace energy-system \
  --set auth.username=admin \
  --set auth.password=secretpass

# MongoDB installieren
helm install mongodb bitnami/mongodb \
  --namespace energy-system \
  # We use the secure images for MongoDB - also to support a wider
```

```
  # range of architectures (e.g. ARM) - not for production use!
  --set global.security.allowInsecureImages=true \
  --set image.repository=bitnamisecure/mongodb \
  --set image.tag=latest --set auth.enabled=true \
  --set auth.rootPassword=secretpass
```

4.3.5. Port-Forwarding für externen Zugriff

```
# RabbitMQ Management UI verfügbar machen (http://localhost:15672)
kubectl port-forward -n energy-system svc/rabbitmq 15672:15672 &

# RabbitMQ AMQP Port für lokale Entwicklung
kubectl port-forward -n energy-system svc/rabbitmq 5672:5672 &

# MongoDB für lokale Entwicklung
kubectl port-forward -n energy-system svc/mongodb 27017:27017 &

# Credentials:
# RabbitMQ: admin/secretpass
# MongoDB: root/secretpass
```

4.3.6. Deliverable

Erstellen Sie einen Abschnitt "Setup-Dokumentation" (1 Seite) mit:

- Screenshot der laufenden k3s Nodes
- Installierte Services Liste
- Aufgetretene Probleme und Lösungen

Bewahren Sie Ihre ggf. erstellten Setup-Skripte und Konfigurationsdateien in einem GitHub Repository auf, wir werden diese in den nächsten Laborübungen erweitern.

5. Abgabe

5.1. Format

Ein zusammenhängendes Dokument (PDF oder AsciiDoc) mit folgender Struktur:

1. **Titelblatt** mit Namen der Teammitglieder
2. **Inhaltsverzeichnis**
3. **Domäne und Anforderungen** (2-3 Seiten)
4. **Softwarearchitektur** (4-5 Seiten)
5. **Setup-Dokumentation** (1 Seite)

6. **Anhang**: Referenzen und weiterführende Links

5.2. Präsentation

- Dauer: 10 Minuten pro Team
- Fokus auf Architekturentscheidungen

5.3. Bewertungskriterien

Kriterium	Gewichtung	Beschreibung
Domänenverständnis	30%	Vollständigkeit der Anforderungsanalyse, Verständnis der Energiegemeinschaften
Architekturqualität	40%	Durchdachte Service-Aufteilung, sinnvolle Kommunikationsmuster, Skalierbarkeit
Technische Umsetzung	20%	Funktionierender k3s Cluster, deployete Services
Dokumentation	10%	Klarheit, Struktur, Diagrammqualität

6. Zeitplan

- **Ausgabe**: 26.09.2025
- **Bearbeitungszeit**: 2 Wochen
- **Abgabe**: Donnerstag, 09.10.2025 23:59 Uhr
- **Präsentationen**: Freitag, 10.10.2025

7. Hilfreiche Ressourcen

7.1. Energiegemeinschaften

- E-Control Energiegemeinschaften
- Koordinationsstelle Energiegemeinschaften
- EAG Gesetzestext

7.2. Architektur

- C4 Model für Architekturdiagramme
- Microservice Patterns
- Martin Fowler’s Microservices Articles

7.3. Technologie

- WSL2 Installation
- k3d Dokumentation
- RabbitMQ Tutorials
- MongoDB Dokumentation
- .NET 9 Dokumentation
- Helm Installation

7.4. Tools für Diagramme

- draw.io
- PlantUML
- Mermaid

8. Hinweise

- Arbeiten Sie in Teams von 2-3 Personen
- Nutzen Sie GitHub für die Versionsverwaltung Ihrer Dokumente
- Beginnen Sie frühzeitig mit der k3s Installation, da hier Probleme auftreten können
- Die Architektur soll realistisch, aber für den Laborumfang angemessen sein
- Denken Sie an die unterschiedlichen Perspektiven (Industrial Informatics vs. Cyber Security)

9. Kontakt

Bei Fragen wenden Sie sich an die Lehrveranstaltungsleitung während der Übungsstunden oder per E-Mail.