Anforderungsanalyse

Es soll für ein Klient einen Kraftwerkssimulation geliefert werden. Das Kraftwerk besteht aus Erzeugern und Verbrauchern.

Funktionale-Anforderungen

- Erzeugern haben eine bestimmte menge von energie ,die erzeugt werden kann, und die Verbraucher haben eine gewisse energiemenge die gebraucht ist.
- Es müssen mehrere Prozesse zusammenarbeiten und kommunizieren können, mittels TCP , UDP RPC und später MQTT.
- Erzeuger/Verbrauche schicken ihre Daten an die Zentrale.
- HTTP GET unterstützen
- Die Zentrale soll mit hilfe von ein HTTP server informationen austauschen, Es soll fähig sein die komplette HTTP abfrage zu lesen.
- Eingehende HTTP Nachricht bis zur Leerzeile (\r\n\r\n) einlesen.
- Es sollen mehrere GET Request gemacht werden können. Die Verbraucher und Erzeuger laufen weiter.
- Mit hilfe von RPC sollen die gesamte historie Daten an einem externen “Client” gebracht werden.
- Mit Hilfe von RPC komponenten ab- und ausschalten.
- Nachrichten mit MQTT Subscriber und publisher Schicken (Ab aufgabe 4).
- Informationsaustausch soll jetzt mit hilfe von MQTT statt UDP passieren.
- Mehrere virtuelle Kraftwerke sollen miteinander durch RPC und den Externen Client kommunizieren können.
- Externen client soll fähig sein nach der Status einzelne komponent jede Zentralle fragen zukönnen.

Nicht-funktionale Anforderungen

- Die Erzeugern sollten auf veränderungen agiere, (sonniger tage, nicht so sonnig…. Elemente gehen kaputt).
- Jedes Prozess soll in einen eigenen Docker Container laufen und die Container sollen durch Docker Compose verbunden werden.
- Erzeuger haben eine Maximalkapazität.
- Verbrauchern und Erzeugern sollen per UDP mit der Zentrale informationen austauschen.
- Erzeuger/Verbrauche werden als Json string umgewandelt und dann an der Zentrale geschickt.
- Der Json string sollte die folgende informationen enthalten: ID, Art des Teilnehmers , Aktuell vorhanden energiemenge, die zur zeit verbrauchte/erzeugte Energie, Maximum kapazität des Erzeugers, und art der Energieerzeugung.
- Der HTTP Server soll mit verschiedenen Browser funktionieren.
- Jede Zentrale hat den gesamten Status aller Komponente, als Liste.
- Grpc für das RPC service nutzen.
- Entwicklung Zwei RPC Klassen und zwei Proto-files, für participants prozess und für Central prozess .
- Die historie Daten Sind groß , und sollen effizient ausgetauscht werden .
- RPC Mit alle komponenten integrieren um datenaustausch und kontrolle zu ermöglichen.
- RPC connection von External Client und Zentrale durch Port 1234
- RPC connection von Zentrale und Teilnehmer durch port 5678.
- Funktionen zur Steuerung der participants / Teilnehmer Entwickeln.
- Funktionen zum teilen der Historie mit dem Externen Client entwickeln.
- UDP komponenten mit MQTT komponenten wechseln um skalierung zu ermöglichen.
- Zentralwerk muss mit andere Zentralwekre zusammen arbeiten können.
- MQTT mit dem system integrieren.
- Für MQTT soll mosquitto verwendet werden.
- Ein MQTT broker soll definiert und gebaut werden.
- Eine Warteschlange definieren
- Alle Informationen der Publisher (Erzeuger) in eine Warteschlange schicken

- Performance-Vergleiche durchführen zwischen dem vorherigen (UDP) System und dem neuem (MQTT).
- Vorhandene Funktionen sollen erhalten bleiben.
- Mehrere zentren sollen gleichzeitig Funktionieren sollen.
- Alle zentralen sind an den Externen client verbunden.
- Es soll ausfall einzelne Zentralen simuliert werden.

Deployment

Um das richtige und effiziente Deployment des Projektes, entwickeln wir zuerst das Projekt sodass es an einer lokalen Maschine funktioniert. Wir schreiben Tests und lassen es laufen und beobachten ob gewisse bugs passieren. Wenn etwas unerwartetes passiert, fixen wir das Problem. Wenn alles gut funktioniert, erweitern wir unsere Docker Container und Cocker Compose files, damit die neuen Änderungen auch im Docker funktionieren.

Nachdem wir die Docker Compose gebaut haben und sehen dass alles funktioniert, legen wir alle files (Projekt files, dockerfiles und alles erwartet) in eine File im gitlab. Z.b legen wir alles was für aufgabe 2 notwendig ist in ein Ordner der Aufgabe 2 heißt. Dass ermöglicht die weiterentwicklung und das Arbeiten in Iterationen, wo jeder Ordner eine Aufgabe repräsentiert.