



Idejna zasnova in postavitve pilota ekosistema interneta stvari z algoritmičnimi orodji

Navodila za namestitvev

Naziv projekta	Idejna zasnova in postavitve pilota ekosistema interneta stvari z algoritmičnimi orodji
Status dokumenta	Končna verzija
Datum izdelave dokumenta	24. 4. 2024
Datum zadnje spremembe	14. 5. 2024
Verzija dokumenta	1.0
Avtor dokumenta	Marko Bajec, Jernej Cvek, Matjaž Pančur, Gregor Burger, Franc Drobnič, Andrej Kos



Zgodovina dokumenta

<i>Datum</i>	<i>verzija</i>	<i>avtor</i>
24.4.2024	0.1	Jernej Cvek
25.4.2024	1.0	Jernej Cvek

Kazalo vsebine

1	Docker	4
1.1	Sistemske zahteve	4
1.1.1	Testno okolje	4
1.2	Seznam uporabljenih gradnikov	4
1.2.1	Seznam Docker slik	4
1.2.2	Seznam odvisnosti	6
1.2.3	Seznam ranljivosti	6
2	Konfiguracija omrežja	8
2.1	Seznam omrežnih vrat	8
2.2	Postopek namestitve	9
2.2.1	Predpogoji	9
2.2.2	Izvorna koda za namestitev platforme	9
2.2.3	Wildcard DNS zapis	9
2.2.4	Inicializacija platforme	10
2.2.5	Zagon platforme	10
2.2.6	Upravljanje platforme	11
3	Docker Swarm mode	11

1 Docker

1.1 Sistemske zahteve

Sistemske zahteve so odvisne od številnih parametrov, ki vplivajo na potrebo po računskih virih. Med ključne **parametre** spadajo:

1. **število in vrsta naprav**, ki bodo priključene oziroma bodo v komunikaciji s platformo,
2. **obseg podatkov**, ki se bodo izmenjevali s platformo,
3. **intenzivnost komunikacije** (frekvenca pošiljanja/prejemanja podatkov posameznih priključenih IoT naprav),
4. **arhitektura platforme**, to je komponente, ki jo sestavljajo ter odvisnosti med njimi - pomemben vidik, ki bo vplival na sistemske zahteve, je vezan tudi na *K8 okolje*, ki ga bo zagotovil naročnik,
5. **zahtevane nefunkcionalne zahteve** platforme, kot so na primer razpoložljivost, varnost, zanesljivost, propustnost ipd.

Minimalne sistemske zahteve bomo podali, ko bodo znani naštetih parametri.

1.1.1 Testno okolje

V času razvoja je testiranje platforme potekalo na virtualnem računalniku s sledečimi specifikacijami:

- 16 vCPU
- 32 GB RAM
- 300 GB disk
- Ubuntu 20.04 LTS
- Docker 24.0.2

Ker so vsi gradniki kontejnizirani, ne predvidevamo težav tudi v primeru, da bi se uporabil kakšen drug OS.

Za testno okolje predlagamo uporabo računalnika s primerljivimi specifikacijami.

1.2 Seznam uporabljenih gradnikov

V tem poglavju so predstavljeni konkretni gradniki, ki tvorijo platformo.

1.2.1 Seznam Docker slik

Gradnik	Github	Dokumentacija	Licenca	Ustrezno st licence	Docker slika	Verzija
Chirpstack Network Server	repo	docs	licenca	✓	chirpstack/chirpstack-network-server	3
Chirpstack Application Server	repo	docs	licenca	✓	chirpstack/chirpstack-application-server	3
Chirpstack Gateway Bridge	repo	docs	licenca	✓	chirpstack/chirpstack-gateway-bridge	3
Postgres DB 15.x		docs	licenca	✓	postgres	15-alpine
Grafana	repo	docs	licenca	✓	grafana/grafana	10.2.1
IoT Agent Manager	repo		licenca	✓	telefonicaid/iotagent-manager	2.1.0-distros
IoT Agent LoRa	repo	docs	licenca	✓	ioeair/iotagent-lora	1.2.5
IoT Agent Ultralight	repo	docs	licenca	✓	quay.io/fiware/iotagent-ul	2.0.0-distros
IoT Agent OPC-UA	repo	docs	licenca	✓	iotagent4fiware/iotagent-opcua	2.2.0
IoT Agent JSON	repo	docs	licenca	✓	quay.io/fiware/iotagent-json	2.4.2-distros
Kong Gateway	repo	docs	licenca	✓	kong/kong-gateway	3.4.1.1
Keycloak	repo	docs	licenca	✓	bitnami/keycloak	22.0.5
MongoDB	repo	docs	licenca	✓	mongo	4.4
Mosquitto	repo	docs	licenca	✓	eclipse-mosquitto	1.6.14
Orion-LD Context Broker	repo	docs	licenca	✓	fiware/orion-ld	1.4.0
QuantumLeap	repo	docs	licenca	✓	orchestracities/quantumleap	edge ¹
Redis	repo	docs	licenca	✓	bitnami/redis	7.2.2
TimescaleDB	repo	docs	licenca	✓	timescale/timescaledb-ha	

1

orchestracities/quantumleap@sha256:cfec3ebfba1f091ec541c67102581d0f0f315318924fcb6429030930ccf71191

opomba

V končni različici postavitve v Docker okolju se bo uporabila zgolj ena instanca (tj. en Docker vsebnik) podatkovne baze PostgreSQL z najvišjo verzijo, ki je še združljiva z vsemi gradniki. V tej instanci bodo vsebovane podatkovne zbirke vseh gradnikov, ki za svoje delovanje potrebujejo podatkovno bazo.

1.2.2 Seznam odvisnosti

Spodnja tabela prikazuje medsebojne odvisnosti gradnikov.

Gradnik	Odvisnosti
Chirpstack Network Server	Redis, Mosquitto
Chirpstack Application Server	Postgres, Redis, Mosquitto, Chirpstack Network Server
Chirpstack Gateway Bridge	Mosquitto
IoT agent LoRa	MongoDB, Orion-LD Context Broker
IoT agent Ultralight	MongoDB, Mosquitto, Orion-LD Context Broker
IoT agent OPC-UA	MongoDB, Orion-LD Context Broker
IoT agent JSON	MongoDB, Mosquitto, Orion-LD Context Broker
Kong Gateway	Postgres
Keycloak	Postgres
Orion-LD Context Broker	MongoDB
QuantumLeap	TimescaleDB, Redis
Grafana	TimescaleDB

1.2.3 Seznam ranljivosti

V tem podpoglavju je navedeno število programskih ranljivosti za posamične gradnike.

Ranljivosti se lahko pojavijo zaradi programskih napak, nepravilne konfiguracije, slabih varnostnih praks med razvojem ali drugih vzrokov. Programske ranljivosti so napake, pomanjkljivosti ali slabosti v programski opremi, ki omogočajo napadalcem, da jih izkoristijo in pridobijo nepooblaščen dostop, povzročijo izpad sistema, ukradejo podatke ali izvedejo druge škodljive dejavnosti.

Za odkrivanje ranljivosti se uporablja odprtokodni varnostni skener [Trivy](#). Priporočljivo je, da se za aktualno stanje opravi ponovno skeniranje naštetih Docker slik.

Docker slika gradnika	Ranljivosti
chirpstack/chirpstack-network-server:3	Total: 32 (UNKNOWN: 0, LOW: 0, MEDIUM: 14, HIGH: 18, CRITICAL: 0)
chirpstack/chirpstack-application-server:3	Total: 32 (UNKNOWN: 0, LOW: 0, MEDIUM: 14, HIGH: 18, CRITICAL: 0)
chirpstack/chirpstack-gateway-bridge:3	Total: 4 (UNKNOWN: 0, LOW: 0, MEDIUM: 0, HIGH: 4, CRITICAL: 0)

Docker slika gradnika	Ranljivosti
postgres:15-alpine	Total: 26 (UNKNOWN: 0, LOW: 0, MEDIUM: 12, HIGH: 14, CRITICAL: 0)
grafana/grafana:10.2.1	Total: 14 (UNKNOWN: 0, LOW: 0, MEDIUM: 12, HIGH: 2, CRITICAL: 0)
telefonicaiot/iotagent-manager:2.1.0	Total: 33 (UNKNOWN: 0, LOW: 11, MEDIUM: 19, HIGH: 3, CRITICAL: 0)
ioeari/iotagent-lora:1.2.5	Total: 77 (UNKNOWN: 1, LOW: 34, MEDIUM: 11, HIGH: 28, CRITICAL: 3) Total: 46 (UNKNOWN: 0, LOW: 2, MEDIUM: 18, HIGH: 21, CRITICAL: 5) CVE-2022-1664 CVE-2019-12900 CVE-2019-8457 CVE-2021-3807 CVE-2021-44906 CVE-2021-44906 CVE-2023-3696 CVE-2021-28918
quay.io/fiware/iotagent-ul:2.0.0-distroless	Total: 23 (UNKNOWN: 0, LOW: 11, MEDIUM: 8, HIGH: 4, CRITICAL: 0) Total: 9 (UNKNOWN: 0, LOW: 0, MEDIUM: 5, HIGH: 2, CRITICAL: 2) CVE-2023-3696 CVE-2022-0686
iotagent4fiware/iotagent-opcua:2.1.9	Total: 125 (UNKNOWN: 1, LOW: 89, MEDIUM: 10, HIGH: 24, CRITICAL: 1) Total: 4 (UNKNOWN: 0, LOW: 0, MEDIUM: 2, HIGH: 1, CRITICAL: 1) CVE-2019-8457 CVE-2023-3696
quay.io/fiware/iotagent-json:2.4.2-distroless	Total: 33 (UNKNOWN: 0, LOW: 11, MEDIUM: 19, HIGH: 3, CRITICAL: 0)
kong/kong-gateway:3.4.1.1	Total: 71 (UNKNOWN: 0, LOW: 50, MEDIUM: 21, HIGH: 0, CRITICAL: 0)
bitnami/keycloak:22.0.5	Total: 165 (UNKNOWN: 0, LOW: 96, MEDIUM: 33, HIGH: 31, CRITICAL: 5) CVE-2023-23914 CVE-2023-23914 CVE-2019-8457 CVE-2024-24806 CVE-2023-45853
mongo:4.4	Total: 25 (UNKNOWN: 0, LOW: 25, MEDIUM: 0, HIGH: 0, CRITICAL: 0) Total: 4 (UNKNOWN: 0, LOW: 1, MEDIUM: 2, HIGH: 1, CRITICAL: 0)
eclipse-mosquitto:1.6.14	Total: 31 (UNKNOWN: 0, LOW: 0, MEDIUM: 2, HIGH: 25, CRITICAL: 4) CVE-2021-36159 CVE-2021-3711 CVE-2021-3711 CVE-2022-37434
fiware/orion-ld:1.2.1	Total: 381 (UNKNOWN: 0, LOW: 144, MEDIUM: 219, HIGH: 18, CRITICAL: 0)
orchestracities/quantumleap:edge ¹	Total: 74 (UNKNOWN: 0, LOW: 2, MEDIUM: 17, HIGH: 44, CRITICAL: 11) CVE-2021-36159 CVE-2022-22822 CVE-2022-22823 CVE-2022-22824 CVE-2022-23852 CVE-2022-25235

Docker slika gradnika	Ranljivosti
	CVE-2022-25236 CVE-2022-25315 CVE-2021-3711 CVE-2021-3711 CVE-2022-37434 CVE-2022-29361
bitnami/redis:7.2.2	Total: 116 (UNKNOWN: 0, LOW: 81, MEDIUM: 25, HIGH: 8, CRITICAL: 2) CVE-2019-8457 CVE-2023-45853
timescale/timescaledb-ha:pg15-ts2.10	Total: 163 (UNKNOWN: 0, LOW: 52, MEDIUM: 111, HIGH: 0, CRITICAL: 0)

1

orchestracities/quantumleap@sha256:cfec3ebfbalf091ec541c67102581d0f0f315318924fcb6429030930ccf71191

opomba

Tekom razvoja se bodo gradniki posodabljali na najnovejše združljive verzije.

2 Konfiguracija omrežja

To poglavje vsebuje informacije o konfiguraciji omrežja, ki je potrebna za pravilno delovanje platforme.

2.1 Seznam omrežnih vrat

Gradnik	Vrata vsebnika	Vrata gostitelja
Chirpstack Network Server	8000/tcp	
Chirpstack Application Server	8080/tcp, 8081/tcp, 8003/tcp	
Chirpstack Gateway Bridge	1700/udp	
Postgres DB 15.x	5432/tcp	
Grafana	3000/tcp	
IoT agent LoRa	4041/tcp, 4061/tcp	
IoT agent Ultralight	4041/tcp, 4061/tcp, 7896/tcp	
IoT agent OPC-UA	4041/tcp, 9229/tcp	
IoT agent JSON	4041/tcp, 7896/tcp	
IoT agent manager	8082/tcp	
Kong Gateway	8000-8004/tcp, 8443-8447/tcp	80 (HTTP) ¹ , 443 (HTTPS), 1700 (UDP - Chirpstack GB), 1883 (MQTT) ²
Keycloak	8080/tcp, 8443/tcp	
MongoDB	27017/tcp	

Gradnik	Vrata vsebnika	Vrata gostitelja
Mosquitto	1883/tcp	
Orion-LD Context Broker	1026/tcp	
QuantumLeap	8668/tcp	
Redis	6379/tcp	
TimescaleDB	5432/tcp, 8008/tcp, 8081/tcp	

¹ Vrata 80 za HTTP se uporabljajo za [Let's Encrypt HTTP-01 challenge](#). Ob uporabi [DNS-01 challenge](#) se lahko ta vrata zaprejo.

² Vrata za MQTT se bodo kasneje spremenila iz 1883 (nekriptirana) v 8883 (kriptirana - MQTTS).

2.2 Postopek namestitve

To poglavje opisuje postopek začetne namestitve platforme na gostiteljski računalnik z uporabo Docker vsebnikov.

2.2.1 Predpogoji

Pred namestitvijo platforme je potrebno zagotoviti:

- gostiteljski računalnik z operacijskim sistem Ubuntu 20.04
- ustrezno nastavljen požarni zid
- javni IP naslov in domena z možnostjo urejanja DNS zapisov
- Docker 24.0.5
- [docker-compose v2.20.3](#)
- Python >= 3.6
- Python [pip](#)
- Python [virtualenv](#)

2.2.2 Izvorna koda za namestitev platforme

Izvorna koda za namestitev platforme se nahaja v Git skladišču.

```
git clone git@github.com:mju-iot/mju-iot-platform.git
```

2.2.3 Wildcard DNS zapis

DNS poskrbi za mapiranje domene v IP naslov gostiteljskega računalnika.

Priporočljivo je uporabiti t.i. "wildcard" DNS zapis.

Tip DNS zapisa	Ime	IPv4 naslov
A	*	192.0.2.1

Zgornji primer bo preslikal domeno in vse njene poddomene (*) na podan IP naslov.

informacija

DNS zapis je potrebno dodati le v primeru produkcijskega načina uporabe.

2.2.4 Inicializacija platforme

Pred prvim zagonom platforme je potrebno izvesti inicializacijo.

Ob zagonu skripte `init.sh` se bo izvedel avtomatski postopek inicializacije. V korenskem direktoriju projekta je potrebno zagnati sledeča ukaza:

```
#      Nastavitev      pravic      za      zagon      skripte
chmod      u+x      init      skripte
#      Zagon
./init.sh
```

Skripta uporabnika vodi čez postopek inicializacije platforme in med drugim poskrbi za:

- izbiro načina namestitve (`Development`, `Production`)
- [`Production`] nastavitve domene in admin e-poštnega naslova
- shranjevanje nastavitev v `settings.yml`
- ustvarjanje datoteke z okoljskimi spremenljivkami `.env`
- ustrezno konfiguracijo gradnikov
- [`Development`] dodajanje lokalne domene `mju-iot.local` in ostalih poddomen v `/etc/hosts`
- možnost zagona platforme po uspešno zaključeni inicializaciji

Za razvoj in/ali lokalni preizkus platforme priporočamo uporabo `Development` načina.

informacija

POMEMBNO: Platforma se zaenkrat zažene s privzetimi gesli in drugimi občutljivimi podatki.

Pred produkcijsko uporabo je potrebno z ročno konfiguracijo poskrbeti za ustrezno spremembo gesel.

2.2.5 Zagon platforme

Ob zagonu je s pomočjo `healthcheck-ov` (v `docker-compose.yml`) poskrbljeno, da se gradniki vzpostavijo zaporedno v pravilnem vrstnem redu, tj. najprej podatkovne baze, nato Kong API gateway, itd.

Tekom prvega zagona se avtomatsko opravi tudi inicializacija podatkovnih baz, Kong-a in drugih gradnikov.

Platforma je pripravljena za uporabo, ko imajo vsi gradniki status `healthy`, kar se lahko preveri z ukazom:

```
docker ps
```

2.2.6 Upravljanje platforme

Upravljanje platforme je možno s sledečimi ukazi, ki jih je potrebno zagnati v korenskem direktoriju projekta:

```
# Zagon platforme
docker compose up -d
# Spremljanje vseh dnevniških zapisov
docker compose logs -f --tail=10
# Spremljanje dnevniških zapisov določenega gradnika
docker compose logs -f --tail=1000 kong-gateway
# Začasna ustavitev platforme
docker compose stop
# Ponovni zagon platforme
docker compose restart
# Izbris platforme
docker compose down -v --remove-orphans
```

3 Docker Swarm mode

Zaradi tehničnih omejitev namestitvenega okolja pri naročniku se bo namesto predvidene implementacije na platformi Kubernetes za potencialno večvozliščno namestitev uporabilo Docker Swarm.

Docker Swarm je način uporabe Docker izvajalnega okolja, ki, podobno kot orkestrator Kubernetes, več strežnikov poveže v enotno platformo, na katero se lahko transparentno namešča večje število kontejnerjev. Namestitev in upravljanje Docker Swarm je preprostejše, kot to velja za Kubernetes, vendar pa je tudi sama zmogljivost in razširjenost Docker Swarm slabša. Še posebej opozarjamo na to, da se trenutno v Docker Swarm način uporabe za specifikacijo storitev uporablja specifikacijo Docker Compose V1, ne pa še naslednjo verzijo specifikacije Docker Compose V2. Po našem mnenju tudi to nakazuje, da samo podjetje Docker svojega lastnega Docker Swarm-a ne gleda več kot primarno platformo za orkestracijo, saj recimo v svojem glavnem produktu, IDE okolju za razvijalce Docker Desktop, zelo dobro podpira uporabo "konkurenčne" platforme Kubernetes.

Za pilotno namestitev priporočamo namestitev vseh komponent na (en) samostojen strežnik oz. virtualni računalnik, namenjen samo temu projektu. S tem se omeji območje vpliva pri napakah/hroščih v komponentah (t.i. "blast radius") samo na to aplikacijo, ne pa na celotno platformo. Pilotna namestitev tudi ne predvideva velikega števila uporabnikov, senzorjev in prometa, tako da je tehnično bolj zahtevna namestitev na večvozliščen način v tej fazi projekta manj smiselna.

Za namestitev aplikacijskega sklada se na Docker Swarm uporabi namesto `docker compose` kar ukaz:

```
docker stack deploy --compose-file docker-compose.yaml
```

Na samem strežniku je potrebno prej eksplicitno vklopiti Swarm način uporabe ("Swarm enabled host"), najprej za kontrolne strežnike ("manager node"), nato še za delavne strežnike ("worker nodes"). Če se na takšni Docker Swarm platformi zažene aplikacijski sklad z `docker-compose` ali `docker compose` (compose V2), se aplikacijski sklad namesti klasično, samo na trenutni strežnik, ne na celotno Swarm platformo.

Prof. dr. Marko Bajec
