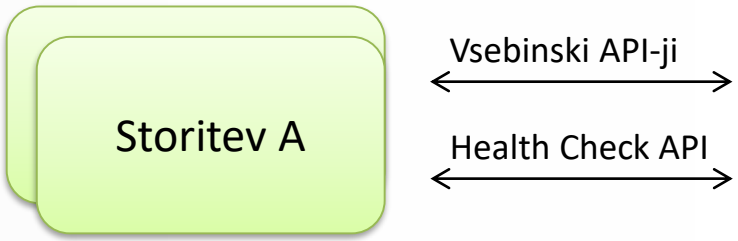
Preverjanje vitalnosti in metrike

Matjaž B. Jurič

API za preverjanje zdravja *(Health Check API)*

- Mikrostoritev se lahko odziva na zahtevke, vendar ne deluje pravilno
  - Npr. nima povezave z bazo
- Mikrostoritev naj implementira končno točko, ki preverja in vrača zdravje mikrostoritve – API za preverjanje zdravja
  - npr. HTTP GET */health*
- Ta mehanizem omogoča zaznavanje nedelujočih instanc mikrostoritev
- API za preverjanje zdravja je lahko periodično klican s strani
  - Mehanizma za odkrivanje storitev
  - Orodja za upravljanje vsebnikov
  - Orodja za monitoring aplikacije
  - Odjemalca za preverjanje zdravja
  - …



API za preverjanje zdravja *(Health Check API)*

- Ob klicu API-ja za preverjanje zdravja mikrostoritev preveri svoje delovanje

- Ob zaznavi nedelujoče instance:

  - Se generira opozorilo *(alert)*

  - Mehanizmi odrivanja storitev odstranijo nedelujoče instance mikrostoritev iz registra

  - Orodja za upravljanje vsebnikov (npr. Kuberneres) nadomestijo bolne stroke *(pod)* z novimi instancami

  - …

Kontrola zdravja *(Health check)*

- Vsak API za preverjanje zdravja implementira več kontrol zdravja
- Instanca mikrostoritve je zdrava, če se vse kontrole zdravja uspešno izvedejo
- Kontrole zdravja delimo na:
  - Standardne
    - Preverjanje povezave do vseh zunanjih storitev (npr. podatkovne baze)
    - Preverjanje zasedenosti trdega diska
    - Preverjanje stanja gostitelja
    - …
  - Po meri
    - Razvijalec sam sprogramira kontrolo zdravja – specifične kontrole zdravja

API za preverjanje zdravja – odgovor

- Mikrostoritev vrača JSON objekt s statusom posameznih kontrol
- Zdrave storitve vračajo HTTP status code 200
- Bolne storitve vračajo status code 503 *service unavailable*

```
{   "outcome": "UP",
    "checks": [
            {
                "name": "DataSourceHealthCheck",
                "state": "UP"
             }
     ]
}
```

Eclipse MicroProfile Health 1.0

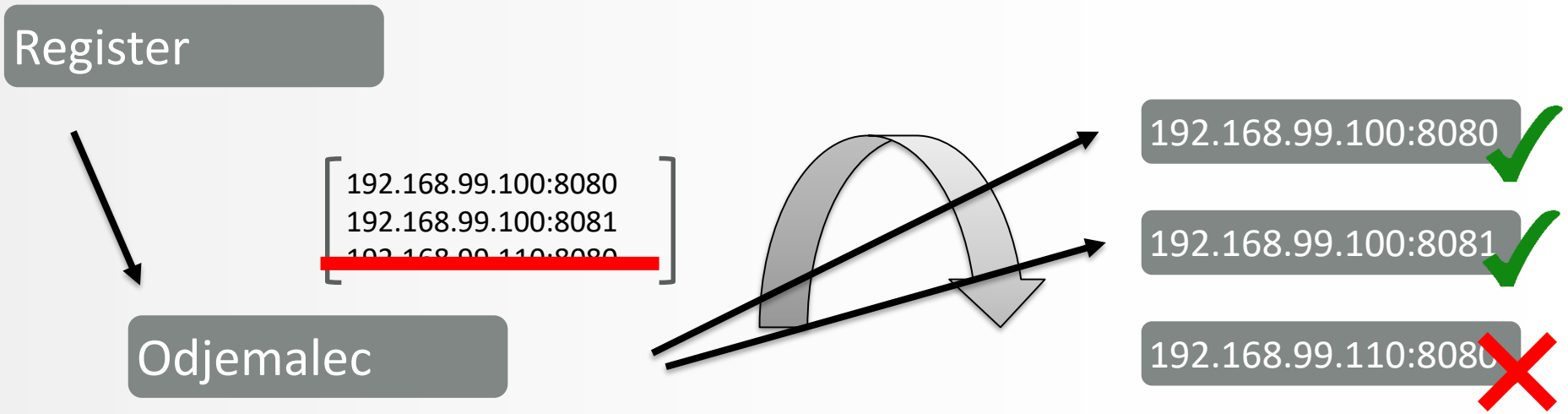
- Del specifikacije MicroProfile 1.2

- Definira
  - Format in protokol API-ja za preverjanje zdravja
  - Java API za definicijo kontrol zdravja po meri

```
@Health
@ApplicationScoped
public class PoljubnaKontrolaZdravja implements HealthCheck {

    public HealthCheckResponse call() {

        [...]

    }
}
```

API za preverjanje zdravja in odkrivanje storitev

- Mehanizem za odkrivanje storitev periodično izvaja klice na API za preverjanje zdravja
- Bolne instance odstrani iz registra storitev
- Pri odkrivanju storitev odjemalcem poda samo naslove zdravih instanc



API za preverjanje zdravja in Kubernetes

- Kubernetes definira sonde življenja *(liveness probes)*
- Sonde življenja preverjajo zdravje strokov
- Sonde življenja definiramo na nivoju stroka
- Izvajajo se periodično v specifciranih časovnih intervalih
- Sonde zdravja lahko zdravje stroka preverjajo s pomočjo
  - Izvajanja poljubnih *bash* ukazov
  - HTTP GET zahtevkov
  - TCP vtičnic *(TCP sockets)*
- Zdrave instance vračajo HTTP *response code* od vključno 200 do manj kot 400

API za preverjanje zdravja in Kubernetes

- Primer konfiguracije sonde življenja

```
livenessProbe:
  httpGet:
    path: /health
    port: 8081
  initialDelaySeconds: 5
  periodSeconds: 3
```

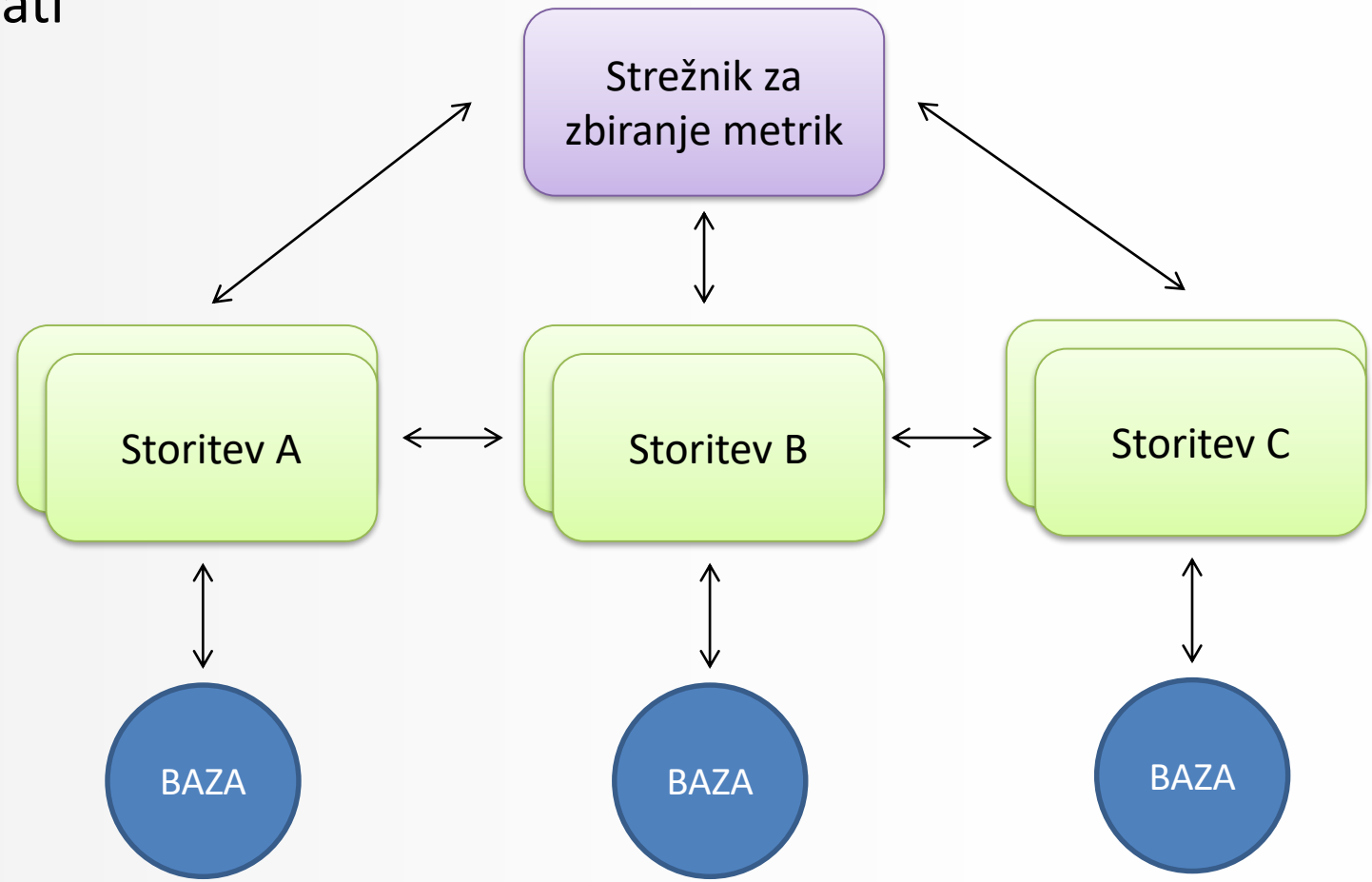
ZBIRANJE METRIK

Metrike

- Metrike so številčne meritve, ki se izvedejo ob določenem času
- Podajo informacijo o trenutnem stanju določene komponente
- Kombinacija večjega števila metrik nudi razumevanje delovanja celotnega sistema
- Metrika je sestavljena iz izmerjene številčne vrednosti in časovne enote
- Večinoma jih zbiramo na določenem intervalu, kot na primer enkrat na sekundo ali minuto

Zbiranje metrik

- Zaradi velike porazdeljenosti aplikacije ni mogoče enovito spremljati aplikacijskih metrik
- Vsaka storitev je odgovorna za zbiranje lastnih metrik ter njihovo posredovanje na centralen strežnik, kjer jih je možno pregledovati in analizirati



Razlogi za zbiranje metrik

- Zbrane metrike so podlaga za skaliranje aplikacij
- Pomagajo pri izboljšanju odpornosti na napake *(fault tolerance)*
- Z zbiranjem metrik lahko zaznamo nepričakovano delovanje posameznih komponent in jih onemogočimo ter tako preprečimo izpad celotne aplikacije
- Pri odstopanju metrik lahko sistem
  - Se samodejno ustrezno odzove
  - Obvesti skrbnika, ki nato ustrezno ukrepa
- Zbiranje metrik na dolgi rok omogoča analizo sistema in podatke o tem, kdaj je aplikacija najbolj obiskana, kje se pojavlja največ napak, povprečen odzivni čas posameznih komponent itd.

Ključne lastnosti zbiranja metrik

- **Celovitost** – Zbiramo metrike
  - Vseh komponent mikrostoritev
  - Okolja, v katerem se izvajajo
  - Vseh storitev in orodij, od katerih so mikrostoritve odvisne
- **Razumljivost**
  - Za metrike mora biti hitro razvidno, katera komponenta ali metoda jih je zabeležila in kaj predstavljajo
  - Imeti morajo priložen čas meritve
- **Granularnost**
  - Zbiramo najbolj nizkonivojske meritve, ki omogočajo točen izvor problema.
- **Konsolidiranost**
  - Vse metrike so združene na enem mestu
  - Omogočamo agregiran vpogled, kar omogoča odkrivanje vseh dejavnikov, ki so vodili do nepričakovanega delovanja aplikacije

Ključne lastnosti zbiranja metrik

- **Konstantnost**
  - Meritve se morajo izvajati neprestano in na kratkih intervalih
  - Sistem zbiranja metrik mora biti robusten in odporen na izpade drugih komponent
- **Učinkovitost**
  - Zbiranje metrik ne sme vplivati na izvajanje merjene komponente
- **Sočasnost**
  - Meritve potekajo v realnem času
  - Sistem nudi sprotno vizualizacijo stanja vseh komponent
  - Nujno je sprotno obveščanje o napakah
- **Arhiviranje**
  - Obvezno je hranjenje meritev na dolgi rok, kar omogoča
    - Vizualizacijo porabe virov in delovanja aplikacije
    - Podrobnejšo analizo in obdelavo meritev
    - Odkrivanje področij, potrebnih optimizacije

Nivoji zbiranja metrik

- Nivo oblaka
  - Delovanje strežnikov
  - Odzivnost glede na geografsko lokacijo strežnikov
  - Odzivnost storitev v oblaku, ki jih uporabljajo mikrostoritve

- Nivo vozlišča
  - Fizične komponente vozlišča (CPU, pomnilnik, disk, omrežje, …)
  - Merjenje povprečne in najvišje zasedenosti fizičnih komponent

Nivoji zbiranja metrik

- Nivo procesov in okolja
  - Koliko CPE, pomnilnika, diska in omrežja zaseda proces
  - Koliko časa teče ter v kakšnem stanju je (se zaganja, teče ali se ustavlja)
  - Metrike izvajalnega okolja (meritve zbiralca smeti *(garbage collector)*, število niti …)

- Nivo aplikacije
  - Metrike specifične za aplikacijo
  - Implementirane na določenih mestih v kodi, kjer vidimo verjetnost napak ali zakasnitev
  - Kategorije
    - Meritve pretočnosti
    - Meritve uspeha
    - Meritve napak
    - Časovne meritve
    - Števci
    - Meritve odvisnih komponent

Eclipse MicroProfile Metrics 1.0

- Del specifikacije MicroProfile 1.2

- Definira
  - Format za izvoz metrik
  - Java API za definicijo aplikacijskih metrik

```
@DELETE
@Path("{customerId}")
@Metered(name = "customer_deleting_meter")
public Response deleteCustomer(@PathParam("customerId") int customerId) {
    Database.deleteCustomer(customerId);
    return Response.noContent().build();
}
```

Kubernetes in Heapster

- Agregator metrik, namenjen uporabi v okolju Kubernetes
- Izvaja se v obliki stroka in avtomatsko najde vsa vozlišča
- Spremlja naslednje metrike
  - Obremenjenost CPE posameznega vozlišča
  - Zasedenost delovnega pomnilnika posameznega vozlišča
  - Odstotek CPE, ki ga trenutno uporablja posamezen strok
  - Odstotek delovnega pomnilnika, ki ga uporablja posamezen strok
- Zbrane podatke shranjuje v zunanjo bazo, npr. InfluxDB

Samodejno skaliranje v okolju Kubernetes

- Kubernetes omogoča samodejno skaliranje mikrostoritev na podlagi metrik
- S samodejnim skaliranjem zagotovimo, da se naša aplikacija samodejno odziva skladno s številom prejetih zahtevkov
- Samodejno skaliranje privzeto temelji na porabi CPU
- Samodejno skaliranje je mogoče zvezati tudi na lastne metrike

Konfiguracija samodejnega skaliranja

- Podobno kot konfiguriramo namestitvene enote, lahko konfiguriramo ***Horizontal Pod Autoscaler.***
  - Po drugi strani ima **kubectl** zelo priročen ukaz **autoscale**, ki poenostavi konfiguracijo**.**
- *HPA* lahko definiramo za **Deployment**, **ReplicaSet** ali **ReplicationController.**
- Prikažemo obstoječe HPA-je:
  - **kubectl get hpa**
- Podrobnosti HPA-ja**:**
  - **kubectl describe hpa**

Konfiguracija samodejnega skaliranja

- Kreiranje *HPA* za namestitev *(Deployment):*
  - **kubectl autoscale deployment <deployment-name> --min=2 --max=10**
  - Parameter *max* je obvezen.

- Določimo lahko tudi pri kateri porabi cpu se ustvari nova instanca:
  - **kubectl autoscale deployment <deployment-name> --min=2 --max=10 *--cpu-percent=80***

- ***Predpogoj za delovanje je nameščen Heapster monitoring.***

Samodejno skaliranje v okolju Kubernetes

- Skaliranje je mogoče izvesti na podlagi lastnih metrik

- Za implementacijo samodejnega skaliranja na podlagi lastnih metrik moramo
  - Metrike izpostaviti na določeni končni točki
  - Metrike zbrati z orodjem za zbiranje metrik, npr. Prometheus
  - Ustvariti mediatorja v obliki skupine API *(API group)*
  - Konfigurirati horizontalni razmnoževalec strokov

Prometheus

- Orodje za zbiranje metrik
- Periodično kliče končne točke storitev, ki izpostavljajo metrike
- Metrike morajo biti zapisane v ustreznem formatu
- Omogoča kreiranje opozoril na podlagi metrik *(alerts)*
- Ponuja močan jezik za poizvedbo po zbranih metrikah
- Potrebno je podati vse naslove, na katerih naj zbira metrike
  - Dobra integracija z orodjem Kubernetes, ki samo zagotovi naslove glede na konfiguracijo namestitev *(Deployments, Services, …)*

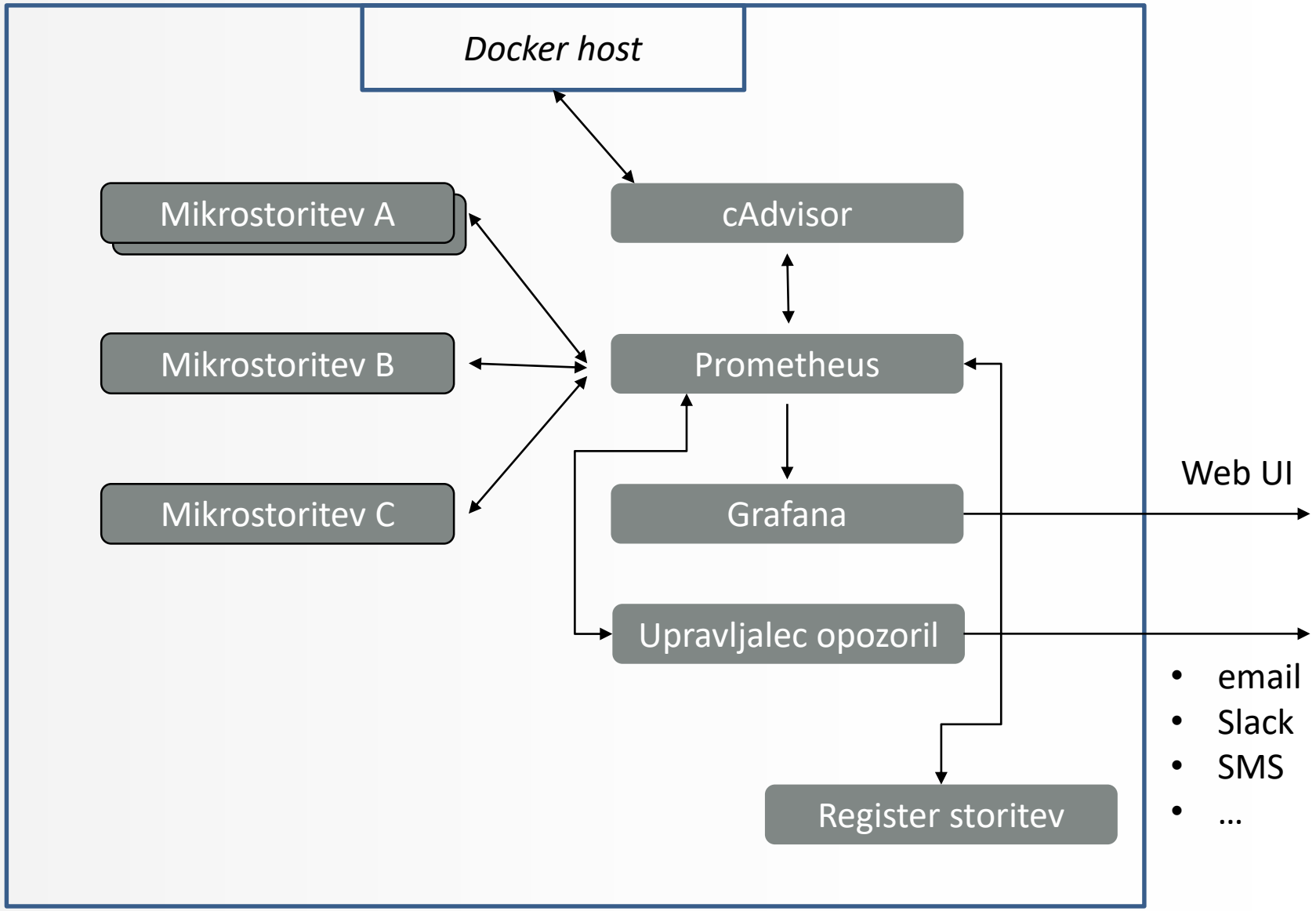
cAdvisor

- Zbira, agregira, procesira in izvaža metrike vsebnikov
- Vključuje preprost uporabniški vmesnik in API za dostop do podatkov
- Zbrane podatke shranjuje v zunanjo bazo (npr. InfluxBD, Prometheus)
- cAdvisor namestimo kot vsebnik na gostitelja (npr. na *Docker host*)
- Zbira podatke o gostitelju in o ostalih zagnanih vsebnikih

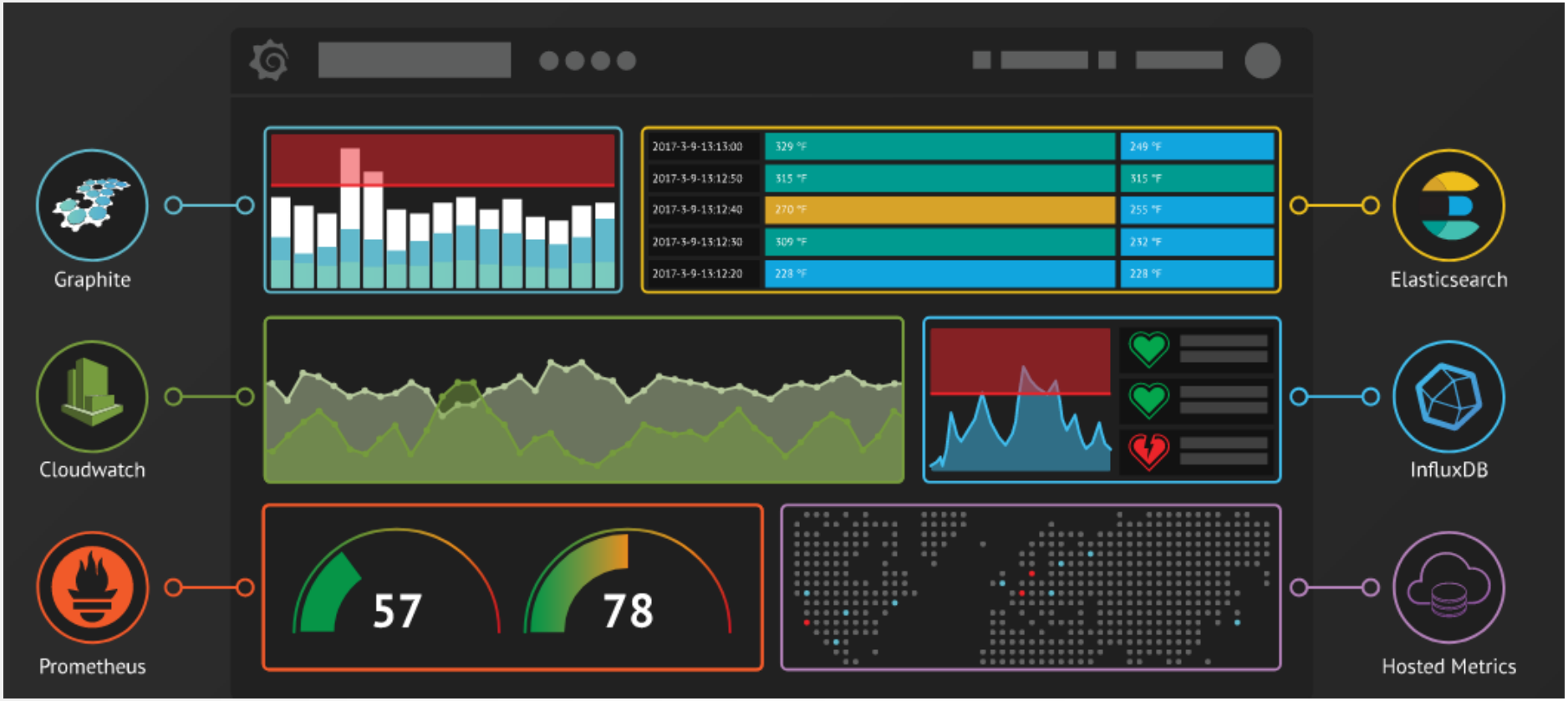
Grafana

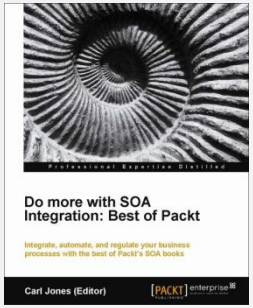
- Odprtokodno orodje za prikaz in analizo metrik
- Omogoča prikaz in združevanje podatkov iz različnih virov
  - Prometheus, Graphite, Cloudwatch, Elasticsearch, InfluxDB, …
- Omogoča definicijo opozoril *(alerts)*
- Omogoča definicijo lastnih grafov in preglednih plošč *(dashboards)*
- Možnost, izvoza, uvoza in deljenja preglednih plošč

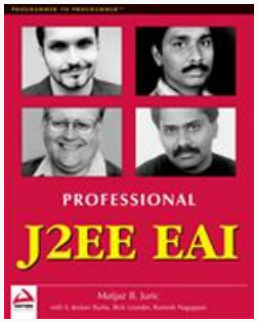
Primer namestitve orodij za zbiranje in analizo metrik

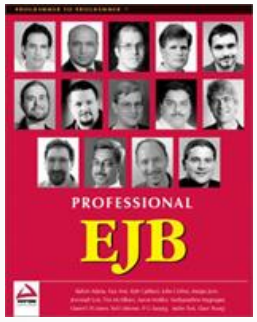


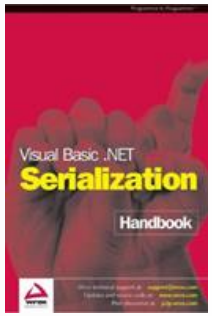
Grafana

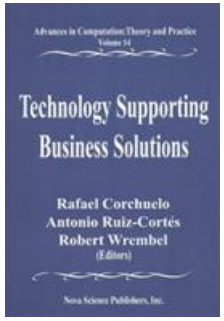

Graphite
Cloudwatch
Prometheus
Elasticsearch
InfluxDB
Hosted Metrics
2017-3-9-13:13:00
329 °F
249 °F
2017-3-9-13:12:50
315 °F
315 °F
2017-3-9-13:12:40
270 °F
255 °F
2017-3-9-13:12:30
309 °F
232 °F
2017-3-9-13:12:20
228 °F
228 °F
57
78

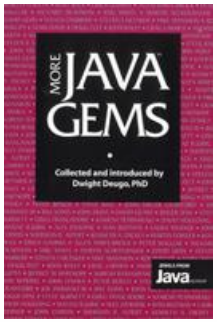












HVALA!

Java Champions · ORACLE ACE Director · IBM CHAMPION · ORACLE Certified Expert · Certified for IBM SOA technology · Certified for IBM CLOUD COMPUTING technology

e-naslov: **http://www.cloud.si**
e-naslov: **http://www.soa.si**
e-pošta: **info@cloud.si**