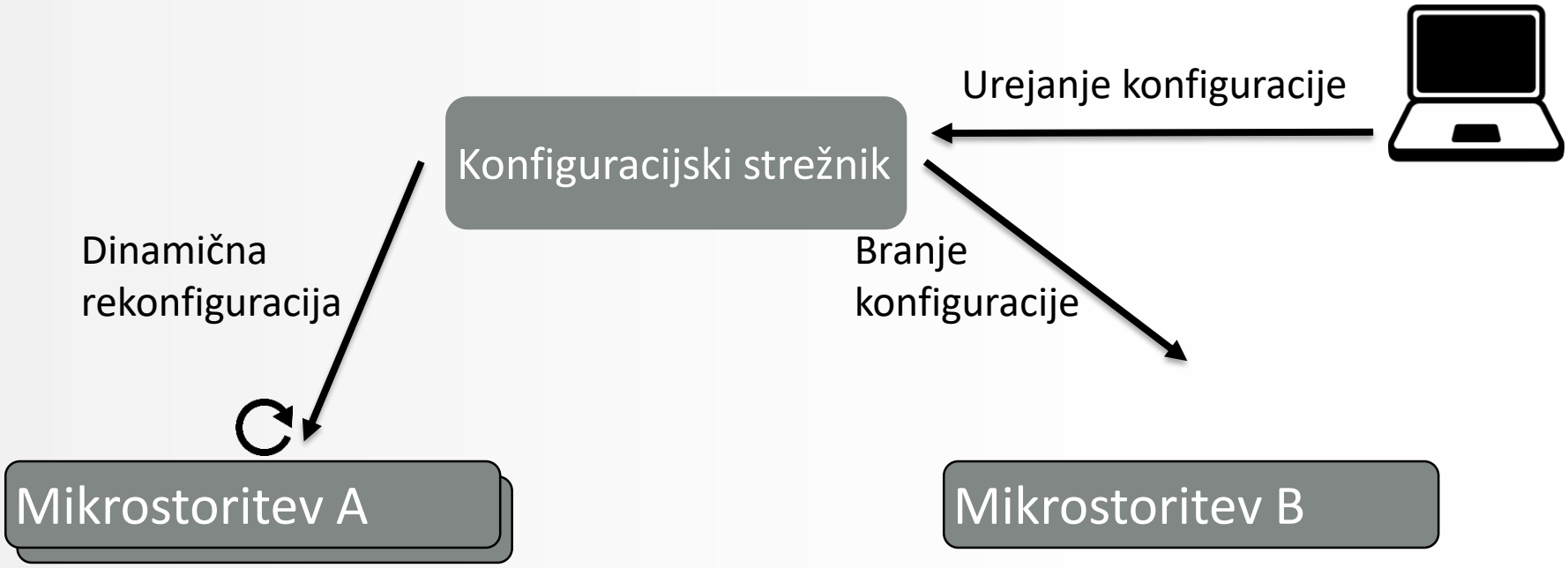
Upravljanje s konfiguracijami

Matjaž B. Jurič

Uvod

- V distribuiranih sistemih igra pomembno vlogo upravljanje s konfiguracijami *(Configuration management)*
  - Želimo drugačno obnašanje storitev v različnih okoljih
  - Konfiguracijo storitve želimo ločiti od implementacije
  - Konfiguracijo storitev želimo spreminjati med njihovim izvajanjem
  - Želimo centralno točko za konfiguracijo vseh instanc storitev



Posebnosti konfiguracije mikrostoritev

- Mikrostoritve potrebujejo konfiguracijo za dostop do infrastrukture in zunanjih storitev (baza, sporočilni sistemi, e-mail…)

- Potrebno je zagotoviti izvajanje mikrostoritve v različnih okoljih brez potrebe po modifikaciji, ponovnem prevajanju in pakiranju
  - Klici na zunanje storitve se lahko razlikujejo med okolji, npr. uporaba različnih podatkovnih baz
  - Vklop in izklop določenih funkcionalnosti storitve v različnih okoljih
  - …

- Konfiguracijo mikrostoritev ločimo od implementacije
  - Definiramo različne vire konfiguracije
  - Mikrostoritev konfiguracijo prebere ob zagonu in spremlja morebitne spremembe konfiguracije med svojim izvajanjem
  - Konfiguracijo za več mikrostoritev ter za vse instance določene mikrostoritve združimo v konfiguracijskem strežniku

Struktura konfiguracijskih vrednosti

- Konfiguracijske vrednosti definiramo v parih

KONFIGURACIJSKI KLJUČ: KONFIGURACIJSKA VREDNOST

- Konfiguracijske ključe lahko gnezdimo

```
kumuluzee:
  name: rso-customers
  env:
    name: dev
  version: 1.0.0
  server:
    base-url: http://localhost:8080
    http:
      port: 8080
```

- Primer konfiguracijske vrednosti: *kumuluzee.env.name: dev*

Viri konfiguracije

- Konfiguracijo shranimo v več virov konfiguracije, tipično uporabimo
  - konfiguracijske datoteke,
  - okoljske spremenljivke ,
  - konfiguracijski strežnik.

- Posamezni viri imajo definirane prioritete. Če je konfiguracijska vrednost za določen konfiguracijski ključ prisotna v več virih, se upošteva vrednost iz vira z najvišjo prioriteto.
  - Možno je definirati privzete konfiguracijske vrednosti npr. v konfiguracijski datoteki, in jih v določenih okoljih prepisati, npr. z uporabo okoljskih spremenljivk.

- Različni viri konfiguracije so avtomatsko združeni v programskem vmesniku za dostop do konfiguracije, ki je na voljo razvijalcu.

Programski vmesniki za dostop do konfiguracije

- Ogrodja za izdelavo mikrostoritev tipično ponujajo programske vmesnike za dostop do konfiguracije

- Programski vmesniku v KumuluzEE
  - *ConfigurationUtil*

```
String environment = ConfigurationUtil.getInstance()
                                      .get("kumuluzee.env.name").get();
```

  - Vstavljanje odvisnosti CDI *(CDI Injection)*

```
@ApplicationScoped
@ConfigBundle("kumuluzee")
public class RestProperties {

    @ConfigValue(value = "env.name")
    private String environment;
}
```

MicroProfile Config

- Specifikacija za konfiguracijo mikrostoritev v profilu *MicroProfile*
- Definira potrebne programske vmesnike za dostop do konfiguracije
- Omogoča dodajanje lastnih virov konfiguracij
- Primer branja konfiguracijske vrednosti

```
@Inject
@ConfigProperty(name = "mp.non-existent-string",
                defaultValue = "Property does not exist!")
private String nonExistentString;
```

Okoljske spremenljivke

- Spremenljivke, definirane na nivoju operacijskega sistema
- Vplivajo na izvajanje procesov
- V arhitekturi mikrostoritev
  - Tipično uporabljene na nivoju vsebnikov
  - Z njimi določimo konfiguracijske parametre, ki so specifični pri posamezni namestitvi mikrostoritve

Okoljske spremenljivke v okolju Docker

- V okolju *Docker* lahko okoljske spremenljivke definiramo na dva načina:

  - V datoteki *dockerfile*

```
ENV <ime> <vrednost>
ENV <ime>=<vrednost> …
```

  - Ob zagonu vsebnika z zastavico *–e* oz. *–env*. V tem primeru prepišemo morebitne vrednosti iz datoteke *dockerfile*.

```
docker run -e POSTGRES_PASSWORD=postgres
                -e POSTGRES_DB=customers -p 5432:5432
                    -d postgres:9.6.3
```

Okoljske spremenljivke v okolju Kubernetes

- V okolju *Kubernetes* lahko okoljske spremenljivke definiramo kot del opisa posamezne namestitve *(deployment)*

```
apiVersion: apps/v1beta1
kind: Deployment
metadata:
  name: order-deployment
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: order
    spec:
      containers:
      - image: nas-repozitorij/orders:0.05
        name: orders
        env:
          - name: KUMULUZEE_DISCOVERY_CLUSTER
            value: openshift
          - name: KUMULUZEE_DISCOVERY_ETCD_HOSTS
            value: http://etcd:2379
          - name: KUMULUZEE_CONFIG_ETCD_HOSTS
            value: http://etcd:2379
          - name: KUMULUZEE_DATASOURCES0_CONNECTIONURL
            value: jdbc:postgresql://postgres-orders:5432/order
```

Konfiguracijske datoteke

- V konfiguracijske datoteke tipično shranimo osnovno konfiguracijo, ki je enaka za vse namestitve, ali pa služi kot *fallback* konfiguracija
- Konfiguracijske datoteke namestimo skupaj z mikrostoritvijo
- Konfiguracijske parametre lahko potem pri vsaki namestitvi prepišemo z drugimi viri konfiguracije z višjo prioriteto, npr:
  - Z okoljskimi spremenljivkami
  - Z uporabo konfiguracijskega strežnika
- Tipično uporabimo zapis v obliki *YAML*
- *V* Javi lahko uporabimo datoteko tipa *Java properties*

Konfiguracijske datoteke

- Primer konfiguracijske datoteke v obliki YAML

```
kumuluzee:
  datasources:
    - jndi-name: jdbc/CustomersDS
      connection-url: jdbc:postgresql://192.168.99.100:5432/customers
      username: postgres
      password: postgres
      max-pool-size: 20
```

- Primer konfiguracijske datoteke v obliki Java properties

```
kumuluzee.datasources[0].jndi-name=jdbc/CustomersDS
kumuluzee.datasources[0].connection-url=
                        jdbc:postgresql://192.168.99.100:5432/customers
kumuluzee.datasources[0].username=postgres
kumuluzee.datasources[0].password=postgres
kumuluzee.datasources[0].max-pool-size=20
```

Konfiguracijski strežniki

- Centralen repozitorij za hranjenje konfiguracije mikrostoritev
- Hrani konfiguracijo za več mikrostoritev
- Tipično realiziran kot distribuirana shramba tipa ključ-vrednost

KONFIGURACIJSKI KLJUČ: KONFIGURACIJSKA VREDNOST

- Konfiguracijo urejajo skrbniki aplikacij
- Mikrostoritve konfiguracijo berejo ob zagonu in med izvajanjem
- Omogočajo rekonfiguracijo mikrostoritve med njenim izvajanjem

Visoka razpoložljivost konfiguracijskih strežnikov

- Konfiguracijske vrednosti so kritični podatki mikrostoritev
- Potrebno je zagotoviti visoko razpoložljivost konfiguracijskih strežnikov
- Konfiguracijske strežnike tipično namestimo v gruči *(cluster)*
- Konfiguracijski podatki so replicirani po vseh vozliščih, kar nam zagotavlja odpornost na izpad posameznega vozlišča
- Za replikacijo podatkov in usklajevanje gruče se uporablja protokol *Raft* in sorodni algoritmi

Protokol *Raft*

- Algoritem za dosego konsenza med vozlišči v distribuiranih sistemih

- Skrbi za
  - Replikacijo podatkov
  - Izbiranje vodje gruče
  - Distribuirano zaklepanje podatkov *(Distributed locks)*

- Vozlišče ima lahko eno izmed treh vlog
  - Vodja *(Leader)* – hrani podatke in skrbi za usklajevanje vozlišč v gruči
  - Sledilec *(Follower)* –hrani podatke in sodeluje pri izbiri vodje
  - Kandidat *(Candidate)* – kandidat za novega vodjo

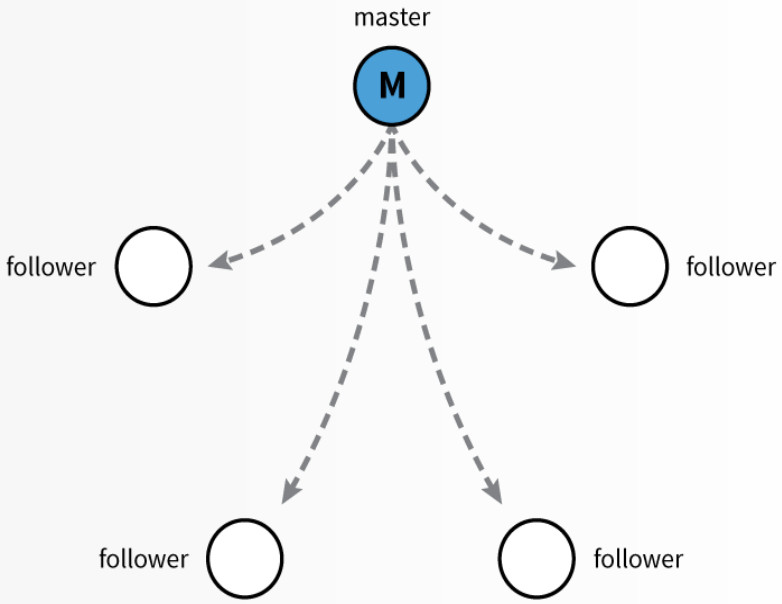
- V primeru izpada vodje gruče, gruča samodejno izvoli novega vodjo

Protokol *Raft*

- Protokol zagotavlja replikacijo podatkov odporno na mrežne particije *(network partitions)*

- Po odpravi mrežne particije protokol poskrbi, da se v vseh vozliščih shranijo pravilni podatki – podatki, ki jih je potrdila večina vozlišč

- Za pravilno delovanje protokola je potrebnih liho število vozlišč



Struktura konfiguracije v konfiguracijskem strežniku

- Isti konfiguracijski strežnik tipično uporabimo za več mikrostoritev in okolij
- Znotraj strežnika je potrebno smiselno strukturirati konfiguracijske parametre
- Tipično se uporablja drevesna struktura
- Konfiguracijske parametre razdelimo glede na posamezne mikrostoritve, verzije mikrostoritev in okolja

Spreminjanje konfiguracije med izvajanjem mikrostoritev

- Če se konfiguracijski parametri spremenijo med izvajanjem mikrostoritve, mora mikrostoritev upoštevati posodobljene konfiguracijske vrednosti

- To je možno doseči na dva načina
  - Pri vsakem branju konfiguracijske vrednosti preberemo vrednost iz konfiguracijskega strežnika -> neučinkovito
  - Z uporabo naročanja na spremembe *(watches)* -> konfiguracijski strežnik ob spremembi vrednosti o tem obvesti mikrostoritev

Koncept naročanja na spremembe *(watch)*

- Z uporabo *watch*-ov konfiguracijski strežnik samodejno obvesti mikrostoritev o spremembi vrednosti. To omogoča
  - Samodejno posodabljanje konfiguracijskih vrednosti znotraj mikrostoritve med izvajanjem
  - Avtomatsko rekonfiguracijo mikrostoritve med izvajanjem
  - Programski odziv na spremembo konfiguracije

Koncept naročanja na spremembe *(watch)* – primeri

- Samodejno posodabljanje vrednosti spremenljivke

```
@ConfigValue(watch = true)
private String stringProperty;
```

- Programski odziv na spremembe v konfiguraciji

```
ConfigurationUtil.getInstance().subscribe(watchedKey,
                                          (String key, String value) -> {

    if (watchedKey.equals(key)) {

        if ("true".equals(value.toLowerCase())) {
            log.info("Maintenence mode enabled.");
        } else {
            log.info("Maintenence mode disabled.");
        }
    }

});
```

Konfiguracijski strežniki - pregled najpomembnejših produktov

- etcd
  - Shramba tipa *ključ – vrednost* za najpomembnejše podate distribuiranih sistemov
- Consul
  - Orodje za odkrivanje in upravljanje s storitvami ter upravljanje s konfiguracijami
- Zookeeper
  - Distribuirana shramba tipa *ključ – vrednost*

etcd

- Visoko-razpoložljiva distribuirana shramba tipa *ključ – vrednost* za najpomembnejše podate distribuiranih sistemov
- Odprtokoden projekt pod okriljem CoreOS
- Uporablja se za
  - Upravljanje s konfiguracijami
  - Podporo odkrivanju storitev
  - Koordiniranje distribuiranih sistemov
- Branje in pisanje vrednosti
  - Z uporabo RESTful API-ja (v2)
  - Z uporabo gRPC API-ja (v3)
  - Z uporabo knjižnic, ki ovijajo API-je

etcd

- Ključi v shrambi so
  - V verziji v2.x organizirani v drevesno strukturo (podobno kot datotečni sistem), npr. */storitve/dnevniki/v1/naslov*
  - Verzija v3.x to opušča, uvaja iskanje po ključih (npr. *prefix*)
- Omogoča naročanje na spremembe vrednosti ključev
  - Ob spremembi je možno obvestiti odjemalce in tako sprožiti rekonfiguracijo
- V orodju Kubernetes uporabljen za interno shranjevanje podatkov

etcd

- Več instanc zagotavlja visoko razpoložljivost sistema

- Uporablja protokol *Raft*
  - Dva tipa vozlišč
    - Master (včasih poimenovan tudi leader) skrbi za koordinacijo followerjev in hrani podatke
    - Follower hrani podatke

  - Protokol zagotavlja
    - Izbiranje novega leaderja
    - Replikacijo podatkov preko vseh vozlišč

  - Priporočljivo je liho število vozlišč (tipično 3 ali 5) zaradi zagotavljanja kvoruma ob odpovedi enega vozlišča

Consul

- Orodje za odkrivanje in upravljanje s storitvami ter upravljanje s konfiguracijami

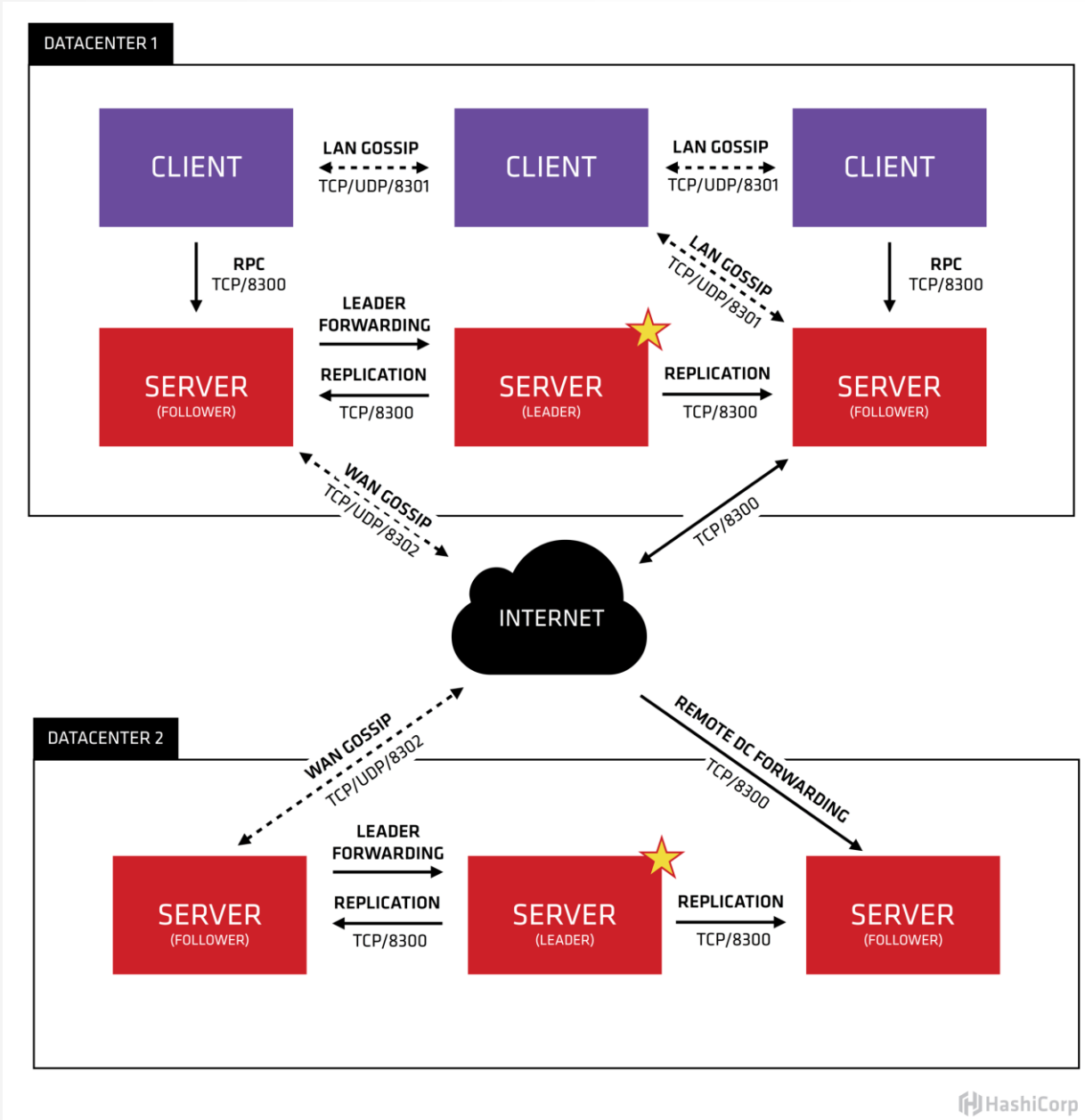
- Konfiguracija je shranjena v podatkovni shrambi tipa *ključ – vrednost*
  - Drevesna struktura
  - Dostop do podatkov z uporabo RESTful API-ja

- Izvaja se v obliki Consul agentov, ki jih delimo na
  - Consul client (odjemalec)
    - Teče na strežnikih, ki gostijo mikrostoritve
    - Neposredno komunicira z mikrostoritvijo
  - Consul server (strežnik)
    - Uporabljajo se za hranjenje podatkov
    - Tipično v gručah 3 do 5 zaradi zagotavljanja visoke razpoložljivosti

Consul

- Distribuirana arhitektura zagotavlja visoko razpoložljivost
- Cluster sestavljajo strežniki in odjemalci
- Uporablja se protokol *Raft*
  - V protokolu sodelujejo samo strežniki
- Odjemalci posredujejo zahtevke strežnikom
- Izberemo lahko enega izmed treh tipov konsistentnosti branj
  - Pisanje je vedno konsistentno

Consul

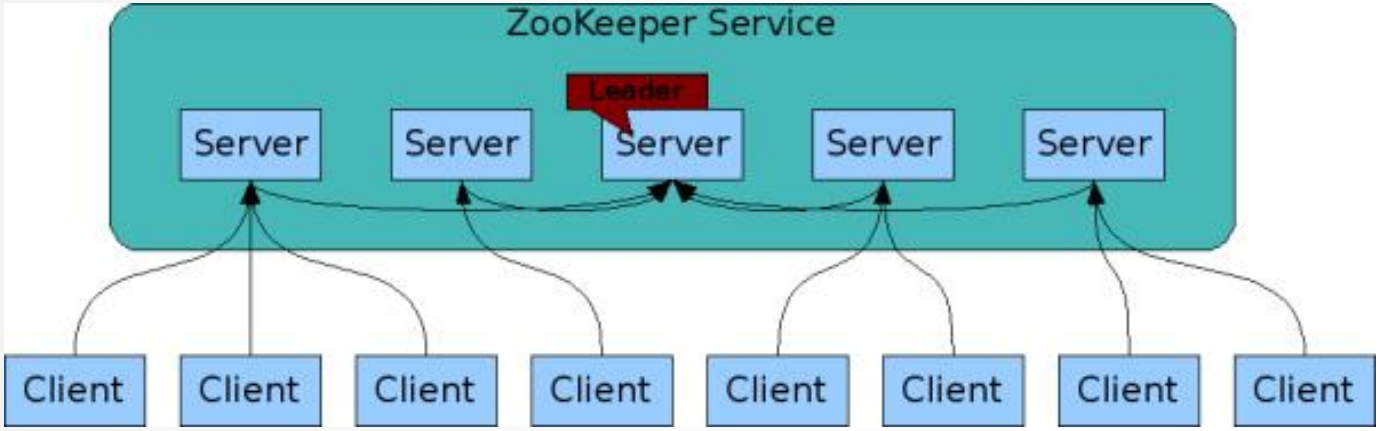

DATACENTER 1
CLIENT
LAN GOSSIP
TCP/UDP/8301
CLIENT
LAN GOSSIP
TCP/UDP/8301
CLIENT
RPC
TCP/8300
LAN GOSSIP
TCP/UDP/8301
RPC
TCP/8300
LEADER
FORWARDING
REPLICATION
TCP/8300
SERVER
(FOLLOWER)
SERVER
(LEADER)
REPLICATION
TCP/8300
SERVER
(FOLLOWER)
WAN GOSSIP
TCP/UDP/8302
TCP/8300
INTERNET
WAN GOSSIP
TCP/UDP/8302
REMOTE DC FORWARDING
TCP/8300
DATACENTER 2
LEADER
FORWARDING
REPLICATION
TCP/8300
SERVER
(FOLLOWER)
SERVER
(LEADER)
REPLICATION
TCP/8300
SERVER
(FOLLOWER)
HashiCorp

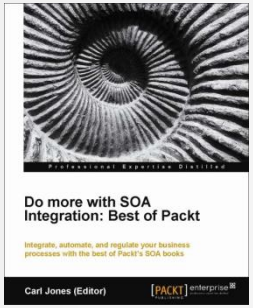
Apache ZooKeeper

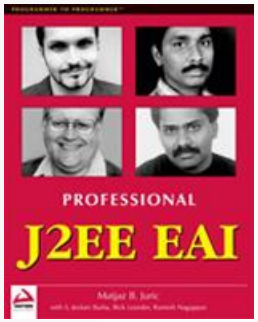
- Odprtokodno orodje za distribuirano, sčasoma konsistentno hierarhično konfiguracijsko shrambo
- Distribuiranim procesom omogoča medsebojno koordinacijo preko deljenih hierarhičnih imenskih prostorov podatkovnih registrov
- Podatki se shranjujejo v podatkovna vozlišča – *znodes*
  - Drevesna struktura
  - Vsako vozlišče lahko hrani podatke in ima podvozlišča
  - Vozlišča lahko hranijo omejeno količino podatkov (1MB sanity check)
- Odjemalske knjižnice Java in C

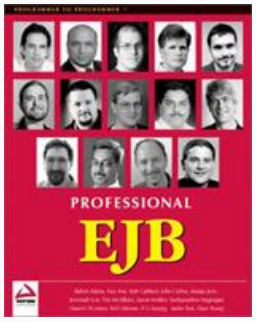
Apache ZooKeeper

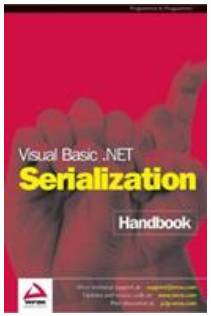
- Distribuirana, sčasoma konsistentna shramba
- Strežniki replicirani znotraj *ensabmle*-a
- Strežniki se morajo zavedati drug drugega
- Podatki so na voljo, dokler je delujočih večina strežnikov
- Uporablja se protokol *Zab*
  - Podobna zasnova kot pri protokolu *Raft*

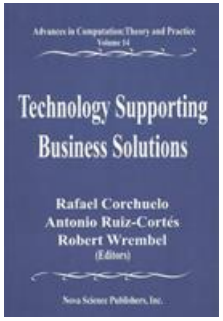


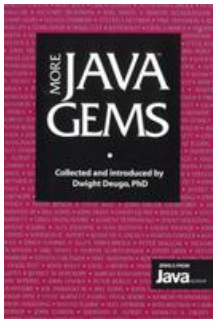












HVALA!

Java Champions

ORACLE ACE Director

IBM CHAMPION

ORACLE Certified Expert

Certified for IBM SOA technology

Certified for IBM CLOUD COMPUTING technology