УНИВЕРЗИТЕТ У НОВОМ САДУ

ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА У НОВОМ САДУ

CLOUD COMPUTING U ELEKTROENERGETSKIM SISTEMIMA

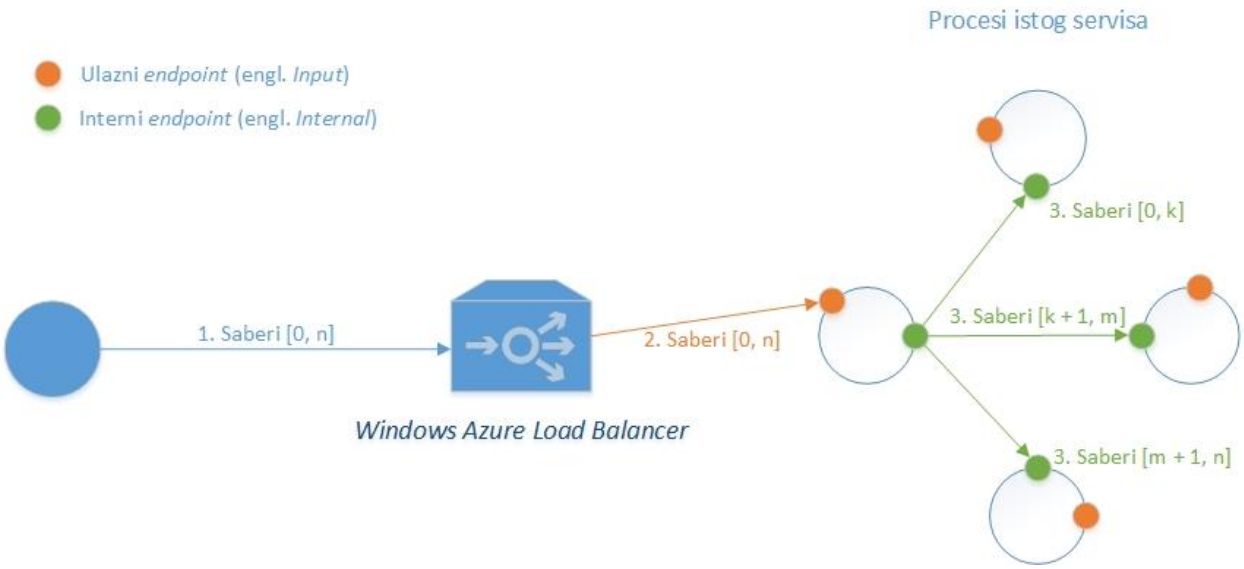
Osnove *Microsoft WindowsAzure* servisa

-SKRIPTA-

Novi Sad, 2020

Vežba 2/3 – *WCF* komunikacija u okviru *Windows Azure* servisa

U vežbi 2 se rešava zadatak koji koristi dve vrste *WCF* komunikacije u *Windows Azure* implementaciji međuservisne komunikacije. Jedna vrsta komunikacije podrazumeva komunikaciju između različitih procesa u okviru istog *Cloud* projekta, i naziva se *inter-role* komunikacija. Druga vrsta komunikacije predstavlja pružanje klasičnog *WCF* servisa. U klasičnom pristupu, zbog distribuiranosti rešenja, zahtev koji je upućen *WCF* servisu prvo stiže do *Windows Azure Load Balancer* komponente. Potom, zahtev se prosleđuje jednom od procesa koji pripadaju datom tipu. Zadatak je izdeljen u dve celine – A i B. Preporuka je da se zadaci rešavaju datim redosledom. Na slici 1 je prikazana arhitektura krajnjeg rešenja zadatka.



Slika 1 Zadatak vežbe 2.

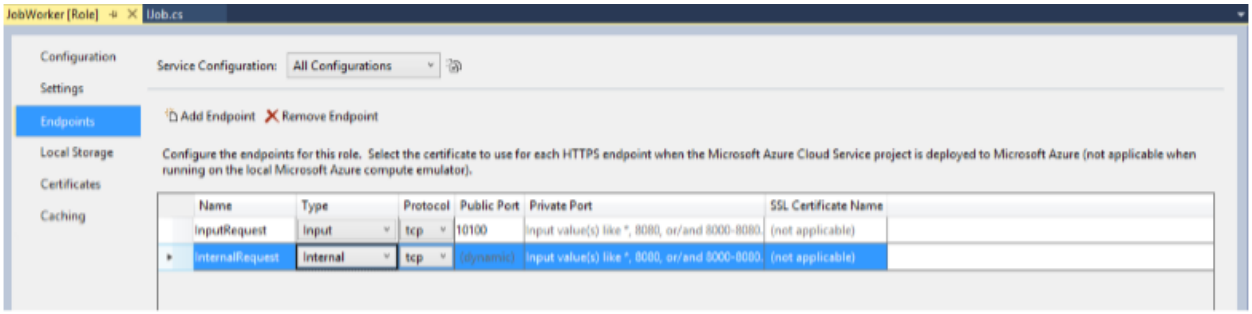
Implementirati distribuirani servis koji sukcesivno sabira brojeve u zadatom intervalu I∈ N_0^+, gde je donja granica intervala uvek 0. Kreirati konzolnu aplikaciju (klijenta) koja će služiti za zadavanje gornje granice intervala. Kreirati *Worker role* tip procesa koji će vršiti sukcesivno sabiranje tako što će koristiti *inter-role* komunikaciju za distribuciju posla na sve raspoložive procese. Servis sukcesivnog sabiranja implementiraju svi procesi u okviru *Worker role* tipa procesa i nije bitno koji proces će primiti zahtev. Nakon primljenog zahteva, proces vrši podelu posla na preostale procese istog tipa procesa. Koristi *inter-role* komunikaciju za kreiranje zahteva za izračunavanje podintervala. Na kraju, vrši se zbir podintervala i zbirna vrednost predstavlja odgovor servisa za sukcesivno sabiranje brojeva.

Pod sabiranjem sukcesivnih brojeva, podrazumeva se sledeći algoritam:

$$f(n) = \sum_{i=1}^{n} i$$

Zadatak B:

Cilj zadatka B je implementacija inter-role komunikacije, označene korakom 3. na slici 1. Prvo je potrebno kreirati enpoint tipa “Internal” koji će se zvati InternalRequest (Slika 2).



Slika 2 Definisanje internal endpoint-a

Kao što možete videti sa slike iznad, sada postoje dva tipa endpoint-a: *Input* i *Internal*. *Input* endpoint je dodat na prošlim vežbama i korišćen je kako bi se kreirao WCF servis za komunikaciju između klijenta (konzolne aplikacije) i servisa (worker role).

Posle dodavanja *Internal* endpoint-a potrebno je podići novi WCF servis koji će da ima contract koji je dat u listingu 3. Pratiti smernice za rešavanje zadatka sa prošlih vežbi, s tim da je za *IPartialJob* servis neophodno da endpoint bude tipa “Internal”. Ovo znači da kako smo na prošlim vežbama imali klasu *JobServer* u ovom slučaju treba napraviti novu, primera radi *PartialJobServer,* i jedina razlika je u tome što je u *JobServer* klasi *externalEndpointName* bio *InputRequest* a sad treba da bude *InternalRequest*.

Listing 3 – IPartialJob interfejs

```
[ServiceContract]
public interface IPartialJob
{
        [OperationContract]
        int DoSum(int from, int to);
}
```

Metodu *IJob* interfejsa kog implementira *JobServerProvider* (napravljeno na prošim vežbama) izmeniti tako da osim ispisivanja intervala u compute emulatoru poziva metode *IPartialJob* interfejsa ostalih instanci prosleđujući iste vrednosti intervala. Za poziv WCF servisa ostalih instanci koji pripadaju istom tipu procesa (*JobWorker WorkerRole*), iskoristiti klasu *RoleEnvironment*. Iz *RoleEnvironment* klase moguće je pristupiti instancama određenog tipa procesa, a tada se može preuzeti i IP adresa za dati *Internal* endpoint. Ovo se radi na sledeći način:

```
            List<EndpointAddress> internalEndpoints = new List<EndpointAddress>();
            foreach (RoleInstance instance in
RoleEnvironment.Roles[RoleEnvironment.CurrentRoleInstance.Role.Name].Instances)
            {
                if(instance.Id != RoleEnvironment.CurrentRoleInstance.Id)
                {
                    internalEndpoints.Add(new
                    EndpointAddress(String.Format("net.tcp://{0}/{1}",

                    instance.InstanceEndpoints[internalEndpointName].IPEndpoint.ToString
                    (), internalEndpointName)));
                }
            }
```

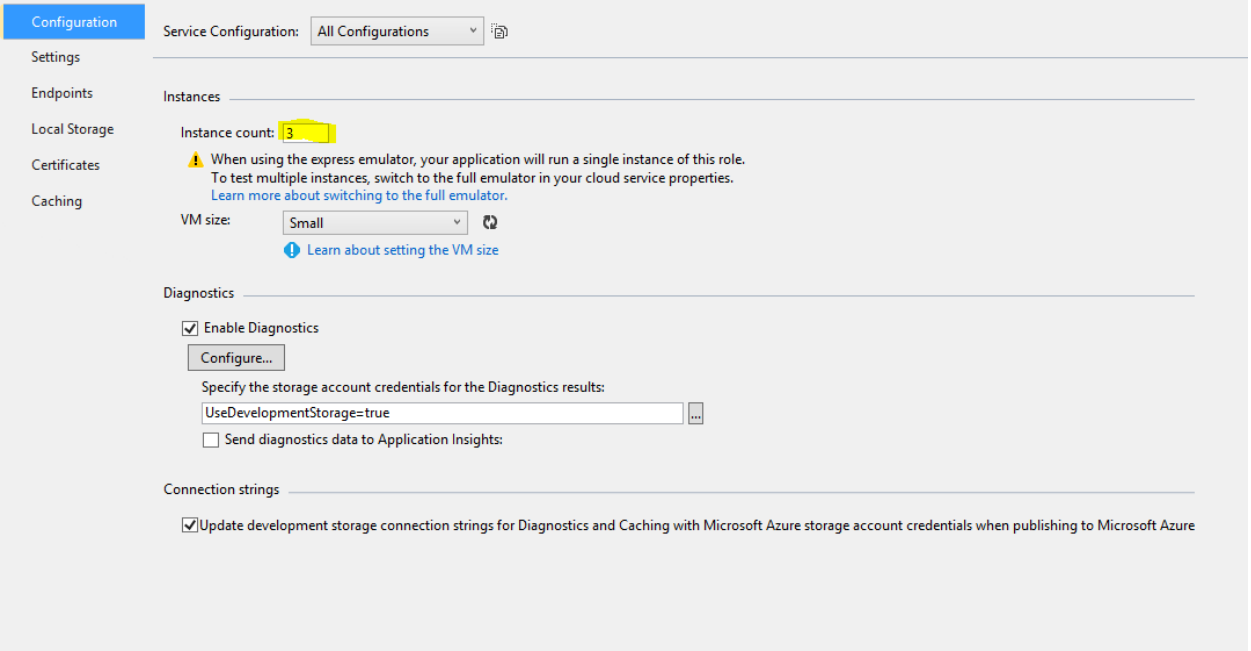
Objašnenje: iz *Roles* uzimamo samo worker rolu od interesea (u ovom slučaju bi radilo I da stavimo kao string ime worker role primer "JobServer"), pa uzimamo sve instance ali samo one koje su različite od trenutne instance koja izvršava ovaj kod.

Sada kada imamo listu *endpoint*-a za svaki process unutar worker role, prolazimo kroz svaku od njih u petlji i pozivamo metodu *DoSum* iz *IPartialJob* interfejsa bratske instance na sledeći način:

```
      IPartialJob proxy = new ChannelFactory<IPartialJob>(binding,
internalEndpoints[index]).CreateChannel();
      proxy.DoSum(openInterval, closeInterval);
```

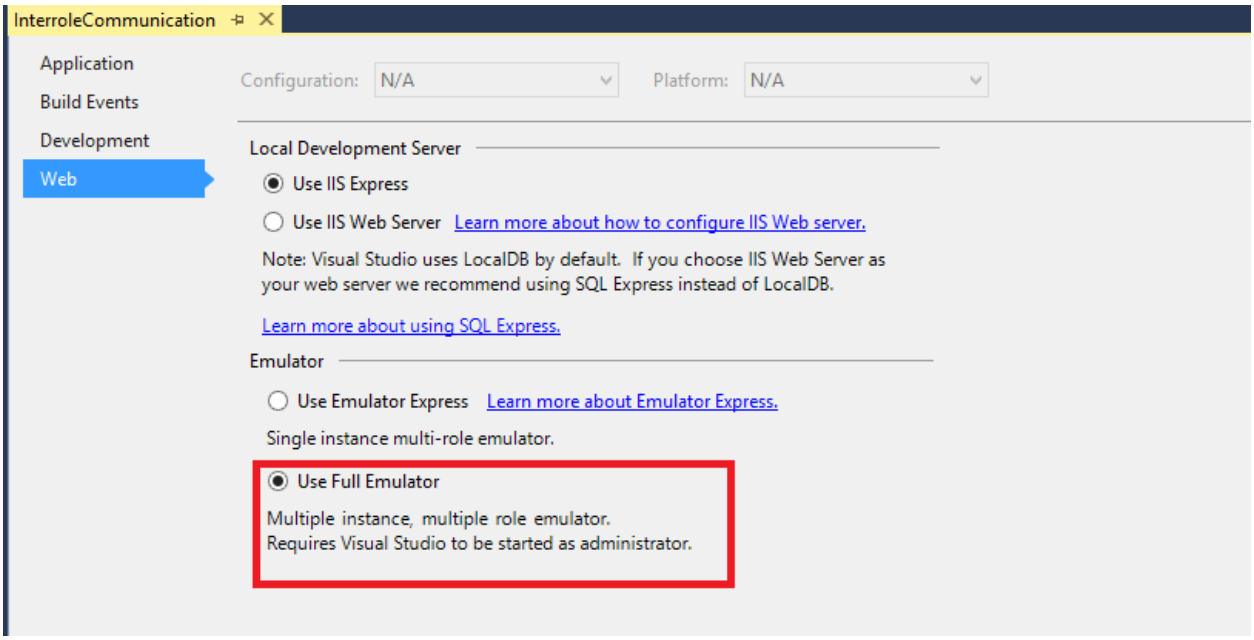
U implementaciji *IPartialJob* interfejsa, ispisivati poruku u compute emulatoru: "DoSum method called - interval [from,to]". Implementirati algoritam sukcesivnog sabiranja i vraćati vrednost kao odgovor metode *DoSum*.

Kako bi postojale više instanci *JobWorker*-a treba podesiti u konfiguraciji *Instance count* na 3. Dvoklikom na *JobWorker* koji se nalazi u projektu *InterroleCommunication*->folder *Roles*, pod stavkom *Configuration* je moguće podesiti *Instance count (*Slika 3*)*.



Slika 3 Instance count

Za rad sa više instanci mora se konfigurisati *Cloud* projekat da radi sa *Full emulator*-om. To se može uraditi desnim klikom na cloud projekat (*InterroleCommunication*) -> Properties -> Web (Slika 4). Takođe, *Visual Studio* mora biti pokrenut kao *Administrator*.



Slika 4 Konfigurisanje Full emulator-a

Dodatno 1:

Cilj zadatka je sukcesivno pozivanje distribuiranih procesa, ali iz zasebnih niti. Ukoliko se sekvencijalno vrši sinhrona WCF komunikacija, distribuiranje posla na procese gubi smisao. Neophodno je vršiti pozive servisa u zasebnim nitima procesa.

Preporuka je da se koristi TPL (engl. Task Parallel Library). Paralelizacija poziva može biti urađena i koristeći Thread klasu iz System.Threading.

Jedan objekat klase Task, koji se nalazi u System.Threading.Tasks, predstavlja jednu nit. Task sadrži metodu Start, koja služi za pokretanje niti. S obzirom na to da jedan Task objekat, u svom konstruktoru očekuje delegat na metodu, može se iskoristiti kreiranje Task objekta iz listinga 1. Za preuzimanje rezultata izvršavanja Task objekta, može se koristiti Result property. Svi zadaci se mogu sinhronizovati uz pomoć statične metode Task.WaitAll.

Listing 1:

```
Task<int> taskObjekat = new Task<int>(() =>
{
       // Ovde implementirati anonimnu metodu - bitno je da vraća int vrednost jer se
       koristi tipizirani Task.
       // Task objekat ne preuzima vrednosti pri kreiranju konstruktora, pa je važno da
       se obrati pažnja na to da se ne koriste deljene, ili globalne promenljive.
 });
```

Dodatno 2:

Cilj zadatka je ravnomerna distribucija posla na preostale procese. Implementirati algoritam ravnomerne raspodele podintervala za preostale procese. Predstaviti interval I kao sumu približno ili tačno ekvidistantnih intervala.

$$I = \sum_{i=0}^{n} (x_i, y_i)$$

pri čemu je zadovoljeno: $|(y_i - x_i) - (y_j - x_j)| < 2, \forall i \in (0, n), \forall j \in (0, n)$, gde je:

n - broj preostalih *Worker role* procesa.