WCF-palvelut & Windows Azure Cloud Service

HAMK
HÄMEEN AMMATTIKORKEAKOULU

Ammattikorkeakoulun opinnäytetyö

Tietojenkäsittely

HAMK Visamäki, kevät 2013

Jarno Niemi

HAMK
HÄMEEN AMMATTIKORKEAKOULU

VISAMÄKI
Tietojenkäsittely
Systeemityö

TIIVISTELMÄ

Tekijä Jarno Niemi **Vuosi** 2013

Työn nimi WCF-palvelut ja Windows Azure Cloud Service

TIIVISTELMÄ

Opinnäytetyön tavoitteena oli tehdä Windows Communication Foundation -palvelu, joka yhdistetään osaksi Windows 8 Store- ja Windows Phone 8 -sovelluksia. WCF-palvelu tullaan isännöimään Microsoftin Windows Azure -pilvialustalla. Työssä esitellään WCF-palvelurajapinnan perusteet, Azuresta työssä käydään läpi pilvipalvelu yleisesti sekä tarkemmin pilvipalvelun yksi osa, Azure Cloud Service ja sen ominaisuudet. Työssä kerrotaan kuinka WCF-palvelu ja Azure otetaan käyttöön ja miksi mikäkin ratkaisu ohjelmistossa on tehty.

Tämän päivän tiedonsiirtotekniikat mahdollistavat palvelusovellusten tuomisen osaksi mobiilisovelluksia. Sovelluksessa ei tarvitse olla itsessään kaikkia toimintoja, vaan voidaan hyödyntää verkkoa ja siellä tarjolla olevia palveluita. Tällöin asiakasohjelmasta voidaan tehdä ”tyhmempi”, kun palvelinsovellus tekee asioita asiakassovelluksen puolesta. Useat mobiilipalvelut toimivat tänä päivänä juuri näin.

Windows Communication Foundation on nykyaikainen ja paljon käytetty tekniikka Web Service -kehitykseen. Microsoftin .NET-alustalla toimiva WCF-palvelu voidaan toteuttaa käyttöjärjestelmä- ja kieliriippumattomaksi, jolloin asiakkaana voi toimia Windows-alustan lisäksi esimerkiksi Java-pohjainen Android-sovellus.

Azure on Microsoft-perheen pilvessä toimiva sovellusalustapalvelu, jonne voidaan luoda muun muassa tietokantoja, internetsivuja, erilaisia palvelusovelluksia, virtuaalikoneita sekä -verkkoja. Pilvipalvelun resurssit voidaan skaalata juuri oikeanlaisiksi. Lisää resursseja saadaan käyttöön parilla hiiren klikkauksella ja kapasiteettia on tarjolla käytännössä rajattomasti. Azuressa yhdistyvät mm. Windows Server, IIS ja SQL Server .NET-alustalla.

Avainsanat Windows Communication Foundation, Azure, Web Service, .NET

Sivut 35 s. + liitteet 11 s.

HAMK
UNIVERSITY OF APPLIED SCIENCES

HAMK Visamäki
Degree Programme in Business Information Technology
System Engineering

Author Jarno Niemi **Year** 2013

Subject of Bachelor's thesis WCF Services and Azure Cloud Service

ABSTRACT

The purpose of this thesis is to develop a network layer for Windows Store 8 and Windows Phone 8's Tic-Tac-Toe application. The network layer will be developed with Windows Azure platform and Windows Communication Foundation. Windows Communication Foundation service will be hosted by Microsoft Windows Azure Cloud Service. This thesis covers the basics of both technologies.

Today's data transfer techniques makes it possible to use services as a part of mobile applications. Local application does not have to have all the features when it can use web services to do part of the work. With this kind of architecture local applications are lighter. Nowadays many applications work like this.

Microsoft .NET frameworks's Windows Communication Foundation is an interoperable solution for developing web services. The client can be a Windows PC using .NET framework and C# or Linux PC using Java or other programming languages.

Windows Azure is Microsoft's platform for cloud computing. Azure offers virtual computers, virtual networks, SQL databases and many other features. Azure's performance can be scaled and it is almost infinite.

Keywords Windows Communication Foundation, Azure, Web Service, .NET

Pages 35 p. + appendices 11 p.

SISÄLLYS

TERMIT JA LYHENTEET

Azure	Microsoftin sovellusalustapalvelu pilvessä, sisältää virtuaalikoneet, web-palvelimet, tietokannat ja erilaiset palvelut
Callback	WCF-palvelun MEP, joka mahdollistaa duplex-tyyppisen tiedonsiirron
Client	Asiakasohjelma, joka käyttää jonkin palvelun tarjoamaa rajapintaa ja sen toimintoja
C#	.NET-sovelluskehyksen tukema ohjelmointikieli
Duplex	Tiedonsiirtoa, jossa viesti kulkee molempiin suuntiin
HTTP	Hyper Trasfer Text Protocol, tiedonsiirtoprotokolla, jota käytetään esimerkiksi www-sivuilla
IIS	Internet Information Service, Microsoftin Windows-ympäristössä toimiva www-palvelin
MEP	Message Exchange Pattern, WCF-kehityksessä käytetty nimitys tiedonsiirtomalleista, jonka pohjalta palvelun toteutus suunnitellaan
SOA	Service Oriented Architecture, palvelukeskeinen sovellusarkkitehtuuri
SOAP	Simple Object Acces Protocol, XML-pohjainen tiedonsiirtotekniikka
SDK	Software Development Kit, kokoelma työkaluja ohjelmistokehitykseen
WCF	Windows Communication Foundation
Web Service	Palvelu, joka tarjoaa asiakasohjelmalla rajapinnan ja erilaisia toiminnollisuuksia
WebSocket	WebSocket tarjoaa duplex-tyyppisen kaksisuuntaisen tiedonsiirtokanavan http-protokollalla
WP8	Windows Phone 8
WS8	Windows Store 8
XML	Extensible Markup Language, käyttöjärjestelmäriippumaton merkintäkieli
.NET	Microsoftin kehittämä ohjelmistokehys ja kehitysympäristö, .NET-ohjelmointikieliä ovat mm. C# ja VB

1 JOHDANTO

Työn tavoitteena on kehittää WCF-palvelurajapinta, joka yhdistetään toimimaan osana Windows 8 Store ja Windows Phone 8 -alustoilla toimivaa Ristinolla-peliä. WCF-palvelu tullaan isännöimään Windows Azure -pilvipalvelussa Cloud Servicenä. WCF-palvelu tarjoaa asiakasohjelmille pelin aikaisen chat-keskustelun sekä välittää Ristinolla-pelin tarvittavat tiedot asiakassovellusten välillä, kuten pelin luonti ja peliin liittyminen sekä siirrot pelilaudalla eli koordinaatit.

Työn yhteistyökumppanina toimii Hämeen ammattikorkeakoulun tietojenkäsittelyn koulutusohjelma. Ohjelmistokokonaisuudesta tehdään koulutusohjelmaa varten ohjelmarunko, jota voidaan hyödyntää opetuksessa ja ohjelmiston jatkokehityksessä. Aihe on ajankohtainen, sillä ohjelmistosuunnittelu on menossa palvelukeskeiseen suuntaa, eli asiakasohjelma saa osan tarvitsemistaan toiminnoista tai koko liiketoimintalogiikan palvelurajapinnalta.

Pilvipalvelut ovat nykypäivää ja Azure on sovellusalustana erittäin kehittynyt ja monipuolinen vaihtoehto siirrettäessä palveluita pilveen. Isännöitäessä palveluita Azurella ei tarvita omaa palvelinkonetta ja konfigurointi on nopeampaa.

2 WEB SERVICE

Web Service on www-sovelluspalvelu, joka tarjoaa esimerkiksi HTTP-protokollan yli toimintoja asiakassovelluksille. Web Service voi tarjota asiakassovellukselle sen tarvitseman liiketoimintalogiikan. Www-sovelluspalvelut pohjautuvat W3C:n standardeihin, joten ne ovat alustariippumattomia. Jos Web Service on toteutettu esimerkiksi Javalla, voi asiakassovellus ottaa palveluun yhteyttä, vaikka se olisi toteutettu PHP-kielellä. (Introduction to Web Services).

2.1 HTTP

HTTP eli Hypertext Transfer Protocol on tiedonsiirtoprotokolla, jota muun muassa web-palvelimet ja www-sivut käyttävät tiedonsiirtoon. Asiakassovellus tai selain lähettää pyynnön ja palvelin vastaa halutulla tavalla, esimerkiksi www-sivulla tai binääridatalla. HTTP:n metodeja ovat esimerkiksi GET, POST, PUT ja DELETE. HTTP-tiedonsiirrossa voidaan käyttää SSL-suojausta (Secure Sockets Layer), jolloin puhutaan HTTPS:stä (Hypertext Transfer Protocol Secure).

2.2 XML

XML eli Extensible Markup Language on merkkauskieli, jota käytetään paljon ohjelmistokehityksen eri yhteyksissä, kuten tiedonsiirrossa, käyttöliittymien toteutuksessa ja tiedon tallennukseen. XML on tietorakenteeltaan

hierarkinen ja rakenteensa ansiosta XML-tiedostot ovat helposti luettavia. XML-merkkaus koostuu elementeistä, jotka voivat sisältää lapsielementtejä, sekä attribuuteista. Esimerkiksi HTML ja XAML ovat XML-pohjaisia merkkauskieliä. XML-merkkausta voidaan määritellä skeemoilla ja nimiavaruuksilla, jotka määrittävät minkälaisia elementtejä, attribuutteja tai muuta sisältöä tiedosto voi sisältää. (Introduction to XML).

Esimerkissä 1 on yksinkertainen XML-tietorakenne, joka sisältää kaksi tuotetta. XML-dokumentin juurielementti on tuotteet, jonka lapsielementtejä ovat tuote-elementit. Tuotteella on attribuuttina id-numero ja sillä on kaksi lapsielementtiä, nimi ja hinta. Hinnalla on attribuuttina valuutta.

```
<?xml version="1.0" encoding="utf-8"?>
<tuotteet>
  <tuote id="1">
    <nimi>Rengas</nimi>
    <hinta valuutta="euro">50</hinta>
  </tuote>
  <tuote id="2">
    <nimi>Lapio</nimi>
    <hinta valuutta="euro">15</hinta>
  </tuote>
</tuotteet>
```

Esimerkki 1. Yksinkertainen XML-tiedosto

2.3 SOAP

SOAP, eli Simple Access Object Protocol on XML-pohjainen tiedonsiirtotapa ja W3C:n standardi. SOAP:ia käytetään tiedonsiirrossa HTTP-protokollan yli.

SOAP-tiedonsiirrossa lähetettävä data on ”kirjekuori” (SOAP envelope), joka sisältää otsikon (header), jossa kerrotaan millainen viesti on kyseessä sekä rungon (body), joka sisältää varsinaisen viestin.

Esimerkissä 2 on kuvitteellinen SOAP-kirjekuori pyynnöstä palvelimelle. Pyynnössä syötetään päivämäärä ja vastauksena voitaisiin saada esimerkiksi kuukausi tekstimuodossa. (SOAP Envelope Element).

```
POST /Testi HTTP/1.1
Host: www.testi.fi
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Body xmlns:m="http://www.testi.fi/kysy">
  <m:GetMonth>
    <m:Date>09.06.2013</m:Date>
  </m:GetMonth>
</soap:Body>

</soap:Envelope>
```

Esimerkki 2. SOAP-kirjekuori

2.4 WSDL

WSDL, eli Web Service Description Language, on XML-muotoinen Web Servicen kuvauskieli. WSDL-tiedosto sisältää kaiken oleellisen asiakkaan tarvitseman tiedon palvelun käytöstä. WSDL kuvaa palvelussa virtaavan datan, palvelun päätepisteet, palvelun tarjoamat toiminnot ja sen viestinvälitysmallin (MEP, Message Exchange Pattern). Esimerkissä 3 on yksinkertainen WSDL-dokumentti. (Introduction to WSDL)

```
<message name="getDataRequest">
  <part name="data" type="xs:string"/>
</message>

<message name="getDataResponse">
  <part name="value" type="xs:string"/>
</message>

<portType name="Test">
  <operation name="getData">
    <input message="getDataRequest"/>
    <output message="getDataResponse"/>
  </operation>
</portType>
```

Esimerkki 3. WSDL-dokumentin perusrakenne

3 WINDOWS COMMUNICATION FOUNDATION

Vuosien saatossa eri laitevalmistajat ovat tehneet paljon tutkimusta hajautettujen toimintojen saralla. Monia tekniikoita on lanseerattu ja yksi viimeisistä on Microsoftin Windows Communication Foundation, eli WCF. WCF, joka tunnettiin alun perin nimellä Indigo, tarjoaa kehittäjille laajat toiminnot tämän päivän ohjelmistokehityksen tarpeisiin. (Pathak 2010, 3)

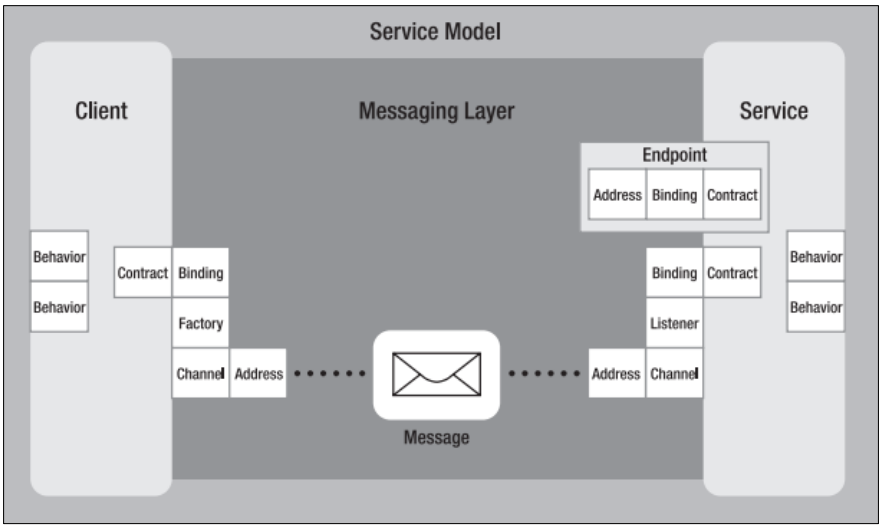
3.1 WCF yleisesti

Hajautettujen systeemien ohjelmistokehitykseen on tarjolla monia työkaluja ja oikean tekniikan valinta työhön voi tuottaa ongelmia. Jokainen tekniikka käyttää omaa ohjelmointimalliaan ja sisältää omat työkalut ja konfiguroinnit. Jos esimerkiksi kehitystyö tehdään .NET-alustalla ja yhtäkkiä huomataan suunnitteluvirhe, jolloin ymmärretään perinteisten XML-pohjaisten Web-Service -rajapintojen olevan parempi vaihtoehto. Tällöin menee koko koodin pohjan uudelleen muokkaamiseen paljon aikaa ja turhaa työtä.

WCF tarjoaa mahdollisuuden ennen itsenäisinä toimineiden teknologioiden yhdistämisen yhdeksi keskitetyksi väyläksi, jolloin samalla tekniikalla voidaan palvelu paljastaa eri protokollia käyttäen asiakkaan käyttöön. Kehitettäessä esimerkiksi talon sisäisiä palveluita, jotka ovat Windows-pohjaisia, voidaan palvelut määrittää käyttämään TCP-protokollaa, jolla ne saadaan toimimaan mahdollisimman nopeasti ja viiveettömästi. Samalla voidaan sama palvelu saattaa käyttöön talon ulkopuolelle HTTP:n ja SOAP:n kautta, jolloin ulkopuolisten asiakkaiden ei tarvitse tukea samaa ohjelmointikieltä tai käyttöjärjestelmää. WCF antaa kehittäjälle mahdollisuuden valita oikea protokolla oikeaan työhön.
(Troelsen, A. 2012, 990)

WCF tukee sekä tavallisia että tyypitettyjä viestejä. Tämä mahdollistaa .NET-alustalla tehokkaan kustomoitujen tyypitysten käytön sekä XML-pohjaisten viestien käytön muilla alustoilla luotujen ohjelmistojen kanssa. WCF tarjoaa monenlaiset sidokset, kuten HTTP, TCP ja MSMQ, jolloin jokaiseen tarpeeseen löytyy oikeanlainen perustus. WCF tukee myös viimeisimpiä Web Service standardeja (WS-*) sekä sisältää kattavat tietoturvaominaisuudet Windows/.NET-alustalla ja Web Service -standardeilla. WCF käyttää System.ServiceModel-nimiavaruutta. Nimiavaruus sisältää kaikki WCF-kehitykseen tarvittavat tyypit.
(Troelsen, A. 2012, 991)

Kuvassa 1 on esitetty WCF-kehityksen yleisarkkitehtuuri, jolla palvelu ja asiakas kommunikoivat. Arkkitehtuurin pääkohdat ovat päätepisteet, joiden avulla kommunikointi tapahtuu. Kuvan pääkohdat käsitellään tulevissa luvuissa.



Kuva 1. WCF-arkkitehtuuri asiakkaan ja palvelun välillä (Pathak 2010, 87)

3.2 SOA

WCF perustuu palvelukeskeiseen arkkitehtuurimalliin, eli SOA:n (Service-Oriented Architechture). SOA on hajautettujen järjestelmien suunnittelutapa, jossa useat autonomiset palvelut toimivat ketjutetusti käyttäen hyvin määriteltyjä rajapintoja välittäen viestejä eri kanavilla. WCF:n tapauksessa luodaan rajapinta, joka määrittelee, mitä osia kutsutaan ulkopuolisen asiakassovelluksen toimesta. (Pathak 2010, 4)

3.3 Sopimukset

Sopimusten ymmärtäminen on WCF-kehityksen avainkohtia, sillä sopimukset määrittelevät WCF-palvelun rungon. WCF-palvelun tärkein sopimus on ServiceContract. ServiceContract tarkoittaa rajapintojen määrittelyä, jolla kuvataan WCF-palvelua. OperationContract-attribuutti määrittää kaikki WCF-palvelurajapinnan käyttämät toiminnot eli metodit. DataContract-attribuutti määrittää WCF-palvelurajapinnan tietotyypit. (Pathak 2010, 64)

Jos ServiceContractissa määritetty rajapinta sisältää vain yksinkertaisia tietotyyppejä (esim. numeeriset, boolean tai string-tyyppiset) voidaan WCF-palvelu luoda käyttämällä pelkästään attribuutteja `[ServiceContract]` ja `[OperationContract]`. Jos palvelu kuitenkin käyttää itse määriteltyjä tyyppejä, tarvitaan käyttöön vielä `[DataMember]` ja `[DataContract]` -attribuutit.

Koodiesimerkissä 4, rajapinta IService1 on WCF-palvelun tarjoama rajapinta. Rajapinta esitellään ServiceContract-attribuutilla ja rajapinnan toiminnot OperationContract-attribuutillla. CompositeType on oma luokka, jota käytetään WCF-rajapinnassa. Luokan tyypittämiseen määritellään DataContract- ja DataMember -attribuutit, koska rajapinnan metodi saa paluuarvoina sekä parametreina luokan olioita. Luokka saa DataContract-attribuutin ja aksessorit DataMember-attribuutin.

```
//ServiceContract-attribuutti määrittelee käytetyn rajapinnan
[ServiceContract]
public interface IService1
{

   //OperationContract-attribuutti määrittelee rajapinnan toiminnot
   [OperationContract]
   string GetData(int value);

   [OperationContract]
   CompositeType GetDataUsingDataContract(CompositeType composite);

   // TODO: Add your service operations here
}

//DataContract-attribuutti määrittää omat tietotyypit
[DataContract]
public class CompositeType
{
    bool boolValue = true;
    string stringValue = "Hello ";

    //DataMember-attribuutti määrittää tietotyyppien aksessorit
    [DataMember]
    public bool BoolValue
    {
            get { return boolValue; }
            set { boolValue = value; }
    }

    [DataMember]
    public string StringValue
    {
            get { return stringValue; }
            set { stringValue = value; }
    }
}
```

Esimerkki 4. Sopimukset

3.4 Päätepisteet

WCF-palveluita suunniteltaessa ja kehitettäessä voidaan puhua A:sta, B:stä ja C:stä. Yhdistelmä ABC tarkoittaa päätepisteen eli endpointin kolmea tärkeintä WCF:n osaa, eli osoitetta (A = address), sidosta (B = binding) ja sopimusta (C = contract). Päätepisteet tarjoavat asiakkaalta pääsyn WCF-palvelun toimintoihin. Isäntä ja asiakas kommunikoivat keskenään hyväksymällä tämän kirjainyhdistelmän. ABC pitää sisällään palvelun peruspilarit, mutta se ei tarkoita sitä, että ensin pitäisi määritellä osoite, sitten vasta

sidos ja lopuksi sopimus. Järjestys on täysin vapaa, mutta yleensä kehitysvaiheessa ensin tehdään palvelun toiminnollisuus eli sopimukset, jotka sisältävät rajapinnan ja sen metodit. Seuraavaksi valitaan käyttötarkoitukseen sopiva sidos ja lopuksi määritellään osoite. Päätepisteet määritellään WCF-projektin Web.config-tiedostoon.

WCF-palvelulla voi olla yksi tai useampi päätepiste. Useammalla päätepisteellä voidaan rajapinta paljastaa useammilla eri tekniikoilla. Tätä tarvitaan silloin, kun asiakassovelluksena on eri alustoilla pyöriviä tai erilaisia verkkotekniikoita käyttäviä sovelluksia. Koodiesimerkissä 5 palvelulle on määritetty kaksi päätepistettä. Esimerkin palvelu on nimeltään Service1 ja se sijaitsee WCFService-nimiavaruudessa. Päätepisteet ovat määritetty käyttämään eri sidoksia, mutta samaa sopimusta. WCF-palvelulla ei voi olla kahta päätepistettä samalla osoitteella.
(Pathak 2010, 21)

```
<services>
   <service name="WCFService.Service1">
       <endpoint address="" binding="basicHttpBinding" contract=
            "WCFService.IService1"/>
       <endpoint address="ws" binding="wsHttpBinding" contract=
            "WCFService.IService1"/>
   </service>
</services>
```

Esimerkki 5. Päätepisteet

3.5 Sidokset

Sidos määrittelee WCF-palvelun tiedonsiirtomekanismin, jota käytetään päätepisteeseen yhdistämisessä sekä keskustellessa päätepisteen kanssa. Sidoksen määrittelemiseen kuuluu tiedonsiirtoprotokollan valinta. Sidoksessa käytetty WCF-palvelun tiedonsiirtokanava määritellään yksisuuntaiseksi, pyyntö-vastaus tai Duplex-malliseksi. Sidoksessa määritellään, kuinka viesti kulkee isännän ja asiakkaan välillä. Viestissä lähetettävä data voidaan enkoodata esimerkiksi XML- tai binäärimuotoon. (Configuring System-Provided Bindings 2012.)

WCF tarjoaa kehittäjälle paljon erilaisia sidosvaihtoehtoja, mutta mikäli yksikään ei tunnu suoraan sopivalta kehittäjän käyttötarpeisiin, voi hän luoda oman sidosvaihtoehdon periyttämällä CustomBinding -tyypin. Taulukossa 1 on esitelty lyhyesti WCF:n valmiiksi tarjoamat sidokset ja niiden ominaisuudet.

Taulukko 1. WCF:n tukemat sidosvaihtoehdot. (Configuring System-Provided Bindings 2012.)

Sidos	***Tarkoitus***
BasicHttpBinding	WCF-projektin oletuksena käytettävä sidos. Käyttää HTTP-protokollaa ja data kulkee oletuksena XML-muodossa. Tukee WS-I Basic Profile 1.1 -ominaisuuksia.
WSHttpBinding	Samantapainen sidos kuin BasicHttpBinding, tarjoaa enemmän Web Service -ominaisuuksia, kuten transaktiot, WS-ReliableMessaging ja WS-Adressing.
WSDualHttpBinding	Toimii kuten WSHttpBinding, mutta on tarkoitettu käytettäväksi silloin, kun palvelun ja asiakkaan välillä viestit kulkevat duplex-mallisesti.
WSFederationHttpBinding	Tukee WS-Federation protokollaa, joka mahdollistaa liitossa olevien organisaatioiden todentamisen ja valtuuttamisen tehokkaasti.
NetHttpBinding	Tarjoaa yhdistelmän BasicHttpBindingistä ja WSDualHttpBindingistä. Sallii sekä pyyntö-vastaus -mallisen yhteyden sekä duplexin (WebSocket) HTTP-protokollalla. Alustana tuettu vain Windows 8.
NetNamedPipeBinding	Tarjoaa turvallisen, luotettavan ja optimoidun käytön samalla koneella kommunikointiin .NET-sovelluksilla.
NetPeerTcpBinding	Tarjoaa turvallisen sidoksen P2P-verkkosovelluksille.
NetTcpBinding	Tarjoaa .NET-sovelluksille turvallisen ja optimoidun sidoksen koneiden välillä kommunikointiin.
MsmqIntegrationBinding	Tarjoaa WCF-palveluille viestin vastaanoton ja lähetyksen olemassa olevaan MSMQ-sovelluksille, jotka käyttävät COM, C++ tai tyyppejä, jotka on System.Messaging-nimiavaruudessa.
NetMsmqBinding	.NET-sovelluksille koneiden väliseen kommunikointiin. Suositeltu MSMQ-sidos.

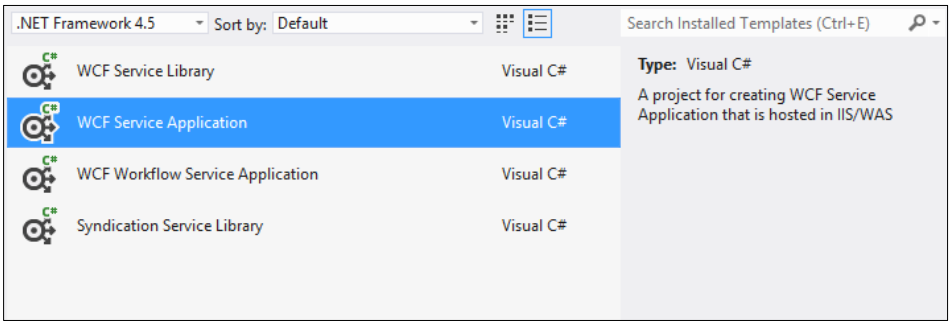
Koodiesimerkissä 6 on määritelty NetHttpBinding-sidos. Sidokselle on määritetty nimi, timeout-määreet, WebSocketin käyttö sekä suojaus on otettu pois päältä. Sidosten määrittely tehdään päätepisteiden tapaan Web.config-tiedostoon.

```
<bindings>
    <netHttpBinding>
        <binding name="netHttp" openTimeout="00:10:00" receiveTi-
            meout="00:10:00" closeTimeout="00:10:00">
            <webSocketSettings transportUsage="WhenDuplex"/>
            <security mode="None" />
        </binding>
    </netHttpBinding>
</bindings>
```

Esimerkki 6. Sidokset

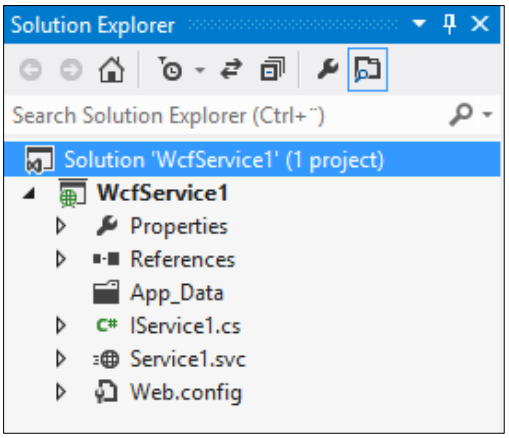
3.6 Palvelun luonti

Visual Studiossa on mahdollista valita neljä erilaista mallipohjaa WCF-palvelua luotaessa. Mallipohjat löytyvät Visual C# -valikosta (kuva 2.), kohdasta WCF. WCF-kehitys on myös mahdollista Visual Basic -kielellä, mutta työssä keskitytään vain kehitykseen C#-kielellä.



Kuva 2. Vaihtoehdot WCF-mallipohjille

WCF Service Library on projektipohja isäntäriippumattomille WCF-luokkakirjastoille. WCF Service Application -mallipohjaa käytetään kehitettäessä esimerkiksi Windowsin Web-palvelimella (Internet Information Service) isännöitävää WCF-palvelua. WCF Workflow Service Application on mallipohja, jolla saadaan käyttöön Windows Workflow Foundation (WF) -työnkulkupohjaiset palvelut. Syndication Service Library -mallipohja tarjoaa tuen syndikaattivirtojen kanssa työskentelyyn, esimerkiksi RSS tai Atom -syötteet. Yleisimmin käytetty projektipohja on WCF Service Application, joka tekee valmiiksi kuvan 3 mukaiset tiedostot.



Kuva 3. Visual Studion luomat tiedostot WCF Service Application -projektipohjalle

IService1.cs on projektin rajapinta, jossa määritellään WCF-palvelulle palvelusopimus eli ServiceContract, metodit ja luokkavakiot. Rajapinnan metodit merkataan `[OperationContract]`-attribuuteillla ja palvelun käyttämät omat luokat `[DataContract]`-attribuutilla sekä luokan aksessorit `[DataMember]`-attribuuteilla. IService1-rajapinnan lähdekoodi löytyy esimerkistä 4.

Service1.svc on palvelurajapinnan (esim. IService1.cs) toteuttava luokka, joka sisältää toteutuksen ja muun palvelun tarvitseman businesslogiikan. Esimerkissä 7 on tehty yksinkertainen toteutus IService1-rajapinnalle.

```
public class Service1 : IService1
{
    public string GetData(int value)
    {
        return string.Format("You entered: {0}", value);
    }

    public CompositeType GetDataUsingDataContract(CompositeType com)
    {
       if (com == null)
        { throw new ArgumentNullException("composite"); }

       if (com.BoolValue)
       { com.StringValue += "Suffix"; }

       return com;
    }
}
```

Esimerkki 7. Rajapinnan toteutus

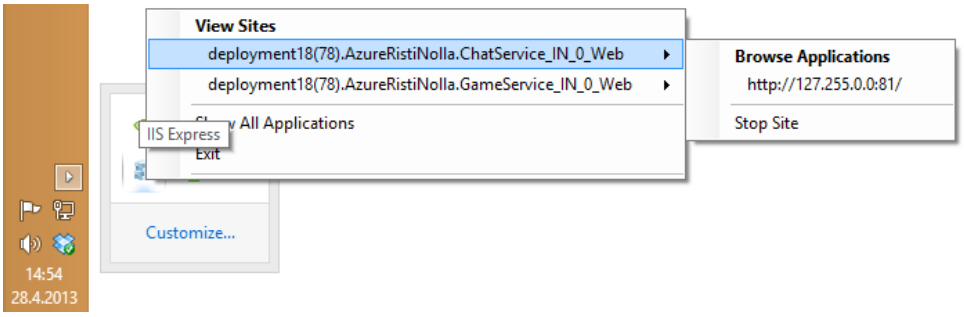
Web.config on XML-tiedosto, joka sisältää mm. WCF-palvelun kaikki määrittelyt, kuten päätepiste, käytetty sidos ja suojaukset. Liitteessä 3 on ChatService-palvelun Web.config-tiedosto.

3.7 Palveluiden isännöinti

WCF-palvelun ajamiseksi tarvitaan alusta, sillä se ei voi pitää itse itseään saatavilla. WCF-palveluiden isännöintiin löytyy kuitenkin monia tapoja.

Työn käytännön osuudessa palvelut isännöidään Azure Cloud Servicenä. Isännöinnin voisi tehdä esimerkiksi Microsoftin Web-palvelimelle, eli Internet Information Servicellä, Windows Process Activation Servicellä (WAS) tai itseisännöintinä esimerkiksi konsolisovelluksella.

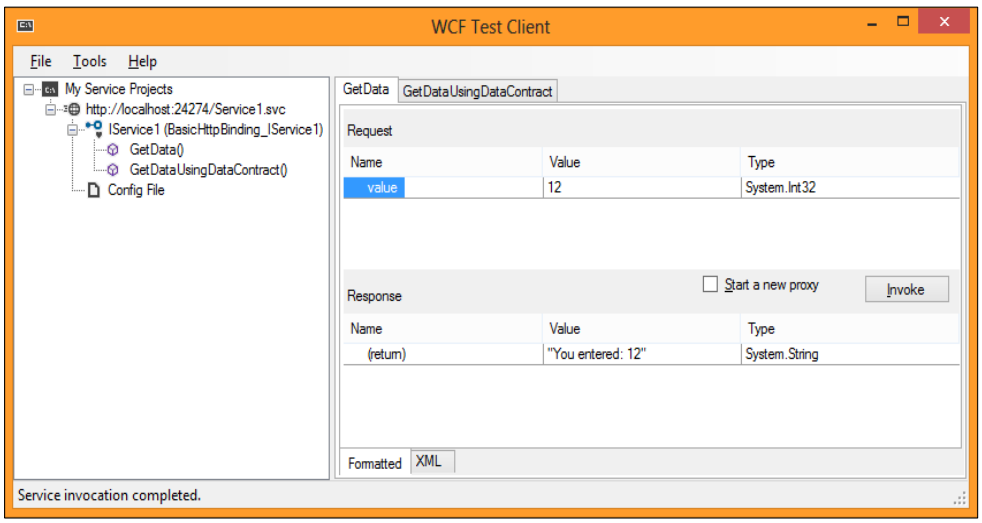
Kehitysympäristössä WCF-palvelu voidaan ajaa yhdellä klikkauksella ilman asetuksiin koskemista IIS Express -palvelimella, joka on integroitu Visual Studioon (kuva 4). IIS Expressillä pärjää kehitysvaiheessa hyvin pitkälle.



Kuva 4. IIS-Express löytyy tehtäväpalkista

3.8 WCF Test Client

Kehittäjän työtä vähentääkseen Visual Studiossa on olemassa WCF-palvelun testaamiseksi WCF Test Client. Sillä voidaan testata palvelun metodien toimintaa ilman, että erikseen luotaisiin oma projekti esimerkiksi konsolisovellus rajapinnan testaamiseksi. Kuvassa 5 on käynnistetty WCF Test Client, käyttöliittymä kertoo rajapinnan tarjoamat metodit. Metodien testausta varten on ikkunan oikea puoli, johon voidaan syöttää metodien parametrit ja tarkastella paluuarvoja. (WCF Test Client 2012).

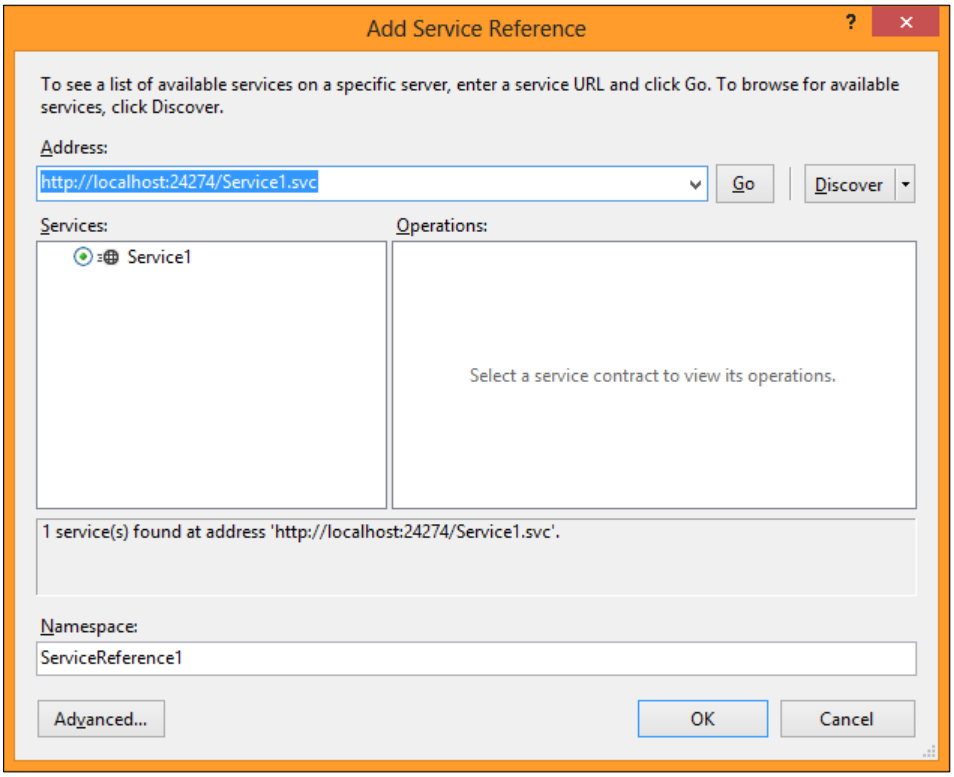


Kuva 5. WCF Test Client korvaa yksinkertaisen konsolisovelluksen testikäytössä

WCF-projektia ajaessa Test Clientin saa päälle klikkaamalla aktiiviseksi WCF-projektista *.svc-tiedoston ja sen jälkeen käynnistämällä projekti Debug-tilaan. Jos projektista on valittuna jokin muu tiedosto, aukeavat projektin tiedostot IIS Expressin virtuaalihakemistoon (selaimeen).

3.9 Palveluiden käyttö

Otettaessa palvelua käyttöön asiakassovellukseen täytyy tietää palvelun sijainti verkossa. Palvelu voidaan ottaa helpoimmillaan käyttöön Visual Studion työkaluilla lisäämällä viittaus palveluun (Service Reference, kuva 6). Viittauksen lisäämiseksi tarvitaan verkko-osoite WCF-palvelun *.svc-tiedostoon. Visual Studio osaa generoida tämän jälkeen palvelun tarjoamat piirteet sopimuksen perusteella luokiksi, joita voidaan käyttää suoraan koodista (kuva 7).



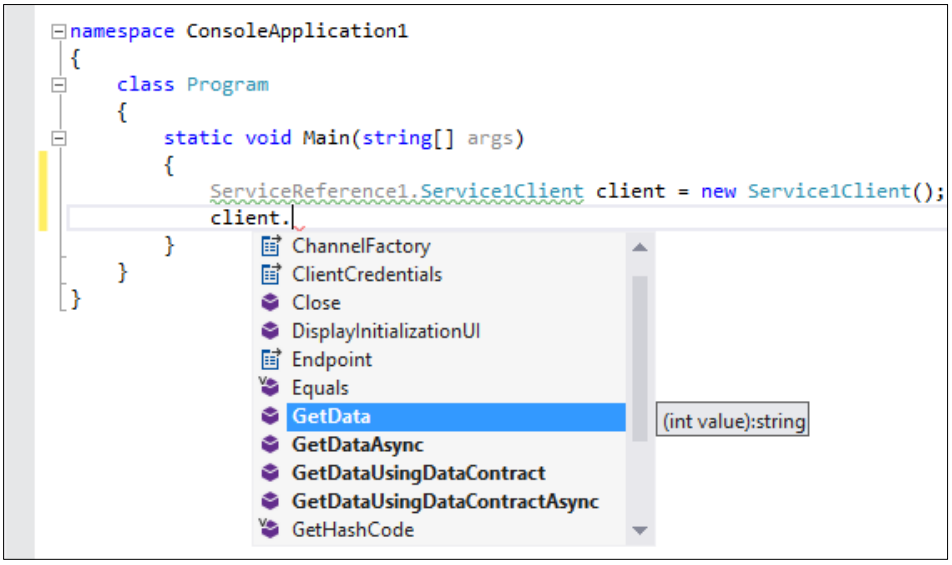
Kuva 6. Viittauksen lisääminen WCF-palveluun, palvelua ei löydy, jos sitä ei osoita palvelimen *.svc-tiedostoon

Esimerkissä 8 on WCF-palvelu otettu käyttöön viittauksen lisäämisen jälkeen. Palvelun käyttöön pitää luoda WCF-palvelusta client-olio, jolla on käytössään kaikki sopimuksen mukaiset toiminnot.

```
//Otetaan palvelurajapinta käyttöön luomalla Service1Client-olio
ServiceReference1.Service1Client client = new Service1Client();

//Rajapinnan GetData-metodi palautuuu string-muotoisena
Console.WriteLine(client.GetData(66));

//DataContract-attribuutilla esitelty luokka voidaan ottaa käyttöön myös
konsolisovelluksessa
ServiceReference1.CompositeType data = new CompositeType
                                        {
                                            StringValue = "testi ",
                                            BoolValue = true
                                        };

//GetDataUsingDataContract-metodi saa sekä parametrina, että paluuar-
vona CompositeType-luokan olion
data = client.GetDataUsingDataContract(data);
Console.WriteLine(data.StringValue);
Console.ReadLine();
```

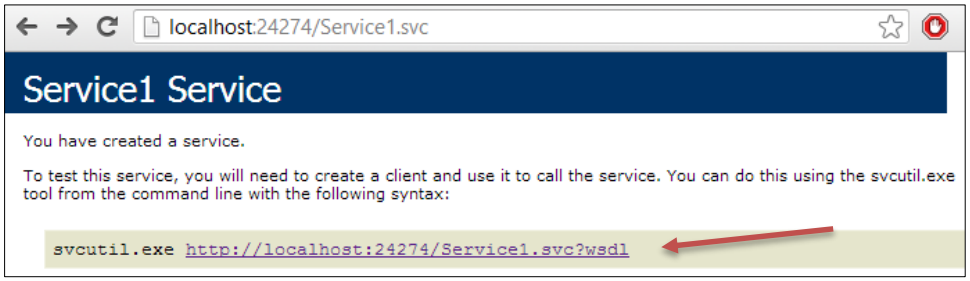
Esimerkki 8. Rajapinnan käyttöönotto konsolisovelluksessa

Viittauksen sopimuksen mukasesti generoima luokka sisältää rajapinnan toiminnot myös asynkronisena (kuva 7). Asynkronisia metodikutsuja käytetään esimerkiksi Windows Store 8 -sovelluksissa.



Kuva 7. Visual Studion IntelliSense osaa tarjota rajapinnan toimintoja

Palvelu voidaan ottaa käyttöön myös ilman viittausta, generoimalla sopimuksen mukaiset luokat asiakkaan käyttöön hyödyntämällä Service Model Metadata Utility Toolia (Svcutil.exe). Isännöidyn WCF-palvelun *.svc-tiedostosta löytyy ohje (kuva 8.), miten palvelu voidaan ottaa käyttöön ChannelFactoryn avulla. Svcutil.exe osaa generoida kehittäjälle sopimuksen automaattisesti. Komentorivillä (Visual Studion Developer Command Prompt) ajetaan kuvan 8 mukainen komento. (How to: Use the ChannelFactory 2012.)



Kuva 8. Paikallisesti isännöity esimerkki WCF-palvelu löytyy tällä kertaa erikoisesta portista, svcutil.exe ei löydy peruskäyttäjän komentoriviltä

Svcutil.exe generoi *.cs-tiedoston, joka sisältää sopimuksen mukaiset luokat ja rajapinnat. Tässä tapauksessa luodaan yksi tiedosto, Service1.cs. Tämä tiedosto voidaan viedä esimerkiksi konsolisovellukseen ja käyttää palvelua ChannelFactoryn avulla esimerkin 9 mukaisesti Kaikki tarvittavat luokat löytyvät WCF:n System.ServiceModel-nimiavaruudesta.

```
//Luodaan sidos sekä päätepiste
BasicHttpBinding myBinding = new BasicHttpBinding();
EndpointAddress myEndpoint = new
            EndpointAddress("http://localhost:24274/Service1.svc");

//Luodaan ChannelFactory-olio, joka saa parametrina sidoksen sekä
päätepisteen
ChannelFactory<IService1> myChannelFactory = new
            ChannelFactory<IService1>(myBinding, myEndpoint);

Otetaan palvelu käyttöön CreateChannel-metodin avulla
IService1 wcfClient1 = myChannelFactory.CreateChannel();
string a = wcfClient1.GetData(39);
Console.WriteLine(a);
```

Esimerkki 9. ChannelFactory

3.10 Duplex

Duplex-tiedonsiirrolla halutaan saada tiedonsiirto toimimaan sekä asiakkaan että palvelimen suuntaan saumattomasti. Esimerkiksi chat-sovellusta suunnitellessa asiakkaan pitää saada lähetettyä viestejä muille käyttäjille ja muiden käyttäjien lähettämät viestit pitää myös saada vastaanotettua. Vaihtoehto tälle olisi käyttää tavallista pyyntö-vastaus -mallia, jolloin asiakassovellus lähettäisi pyyntönä palvelimelle viestin ja palvelin vastaisi lähettämällä takaisin uudet lukemattomat viestit tai ilmoituksen siitä, ettei uusia viestejä ei ole. (Pathak 2010, 64). Asiakassovellus joutuisi myös tietyn ajan

välein kysymään palvelimelta, onko uusia viestejä saatavilla. Tätä tapaa kutsutaan pollaukseksi (Polling). Se olisi vaihtoehtoinen tapa toteuttaa chat-sovellus, mutta pollaus rasittaa palvelinta huomattavasti turhilla pyynnöillä. (DotNet WebSocket Programming 2012)

Muuttamalla asiakassovelluksen ja palvelimen keskustelun duplex-muotoon, asiakassovellus voi lähettää viestinsä palvelimelle, ja palvelin lähettää uudet viestit asiakasohjelmalle juuri silloin kuin viestit ovat tulleet ilman erillistä pyyntöä. Tällöin asiakassovellus ja palvelin käyttävät samaa kanavaa tiedonsiirrossa ja tieto kulkee molempiin suuntiin viiveettömästi.

3.11 MEP

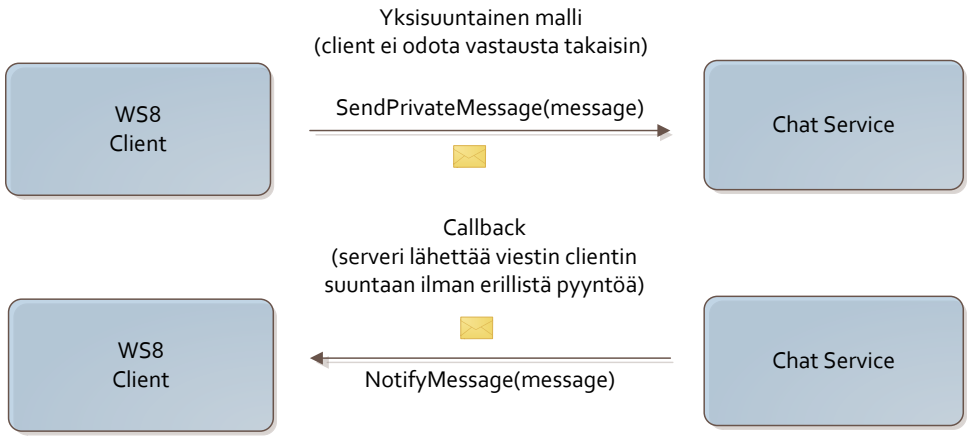
Asiakasohjelman ja palvelimen välillä voidaan käyttää monenlaisia tiedonsiirtomalleja (Message Exchange Patterns). Tavallisin tiedonsiirtomalli asiakasohjelman ja palvelimen välillä on pyyntö-vastaus -malli. Asiakasohjelma lähettää palvelimelle pyynnön (request) ja palvelin palauttaa vastauksen (reply). Normaali www-sivu toimii tällä mallilla; käyttäjä lähettää selaimella palvelimelle pyynnön, eli url-osoitteen ja mahdolliset HTTP-parametrit. Palvelin vastaa selaimelle lähettämällä html-koodia sisältävän www-sivun, kuten kuvassa 9.

Kuva 9. Pyyntö-vastaus -kommunikointimalli

Duplex-tiedonsiirtomallissa tiedonsiirto on kaksisuuntaista. Pyyntö-vastaus -mallissa lähettäjä ja vastaanottaja pysyvät samoina, mutta duplex-mallissa lähettäjä voikin olla myös vastaanottaja. Viesti kulkee siis molempiin suuntiin, kuten kuvan 10 esimerkissä. (Pathak 2010, 63).

Kuva 10. Duplex-malli

Työssä käytännön osuudessa sovellettu malli on duplex, jolloin lähettäjän sekä vastaanottajan roolissa toimii sekä asiakasohjelma (WS8 & WP8) että palvelin (WCF). Tässä mallissa lähetetyt viestit ovat yksisuuntaisia (void), jolloin lähettäjä ei odota, eikä myöskään saa vastauksia. Chat-sovelluksessa lähettäjä lähettää viestin palvelimelle ja palvelin lähettää viestit edelleen muille asiakasohjelmille. Esimerkki kommunikoinnista on kuvassa 11.



Kuva 11. Käytännön osuudessa toteutettu tiedonsiirto, jossa molemmat osa puolet toimivat sekä lähettäjänä että vastaanottajana samalla kanavalla

4 AZURE CLOUD SERVICE

4.1 Yleistä Azuresta

Azure on Microsofin vuonna 2010 lanseeraama pilvipalvelu, jolla voidaan esimerkiksi ajaa virtuaalikoneita tai tehdä virtuaaliverkkoja. Sitä voidaan myös käyttää Web-palvelimena tai tietokantana. Azuressa yhdistyy Windows-palvelin, IIS, SQL-serveri sekä .NET-sovelluskehys ja sillä voidaan ajaa .NET-yhteensopivia koodeja sekä Javaa, Node.js:ää tai esimerkiksi PHP:tä. Azure on nopealla kaistalla ja suurella laskentateholla varustettu, nykypäiväinen vaihtoehto esimerkiksi yrityksille infrastruktuurin ylläpitämiseen, varmuuskopioititarkoituksiin, PaaS (Platform as a service) tai SaaS (Software as a Service) -käyttöön. Azuren avulla infrastruktuurin toimintakyvyn ylläpito sekä varmuuskopiointi ovat Microsoftin hoidossa.

4.1.1 Hallittavuus

Azuren peruskäyttö tapahtuu selaimella, Manage-portaalista voidaan luoda virtuaalikoneet ja -verkot, tietokannat yms. Portaalin kautta voidaan tarkastella omia palveluita ja statistiikkoja esimerkiksi suoritin- tai verkkokäytöstä.

Palveluita voidaan muun muassa käynnistää uudelleen yhdellä hiiren klikkauksella Instances-välilehdeltä. Palvelut voidaan määrittää alueellisesti

esimerkiksi Pohjois-Eurooppaan tai Aasiaan. Suomea lähin Azure-palvelinkeskus löytyy Irlannista. Aluemääritys takaa mahdollisimman viiveettömän käytön.

4.1.2 Skaalattavuus

Azure sopii ympäristönä monenlaisiin tarpeisiin. Tehojen puolesta se on lähes rajaton, sillä muutamalla hiiren klikkauksella voidaan saada lisättyä prosessoriytimiä, koko virtuaalikone luokkaa suurempaan tai lisää koneita. Skaalaus onnistuu sekä ”ylös”, että ”ulos”. Ylös skaalaus tarkoittaa lisää laskentatehoja ja ulos skaalaus tarkoittaa lisää virtuaalikoneita.

Instansseja, eli virtuaalikoneita voidaan määrittää lisää siltä varalta, jos palvelulle tulee käyttöpiikkejä, jolloin kuormaa voidaan tasata kaikille. Kuormantasaaja toimii kehittäjän puolesta ja jakaa samanaikaiset käyttäjät tasaisesti eri koneille. Instansseja on myös hyvä olla siltä varalta, että yksi instanssi palvelusta kaatuu, jolloin käyttäjät siirtyvät automaattisesti toiselle instanssille tai kaatumisen yhteydessä käynnistetään uusi instanssi. (Windows Azure Cloud Services Concepts 2013.)

Instanssien ja virtuaalikoneen määritys onnistuu Cloud Service -projektin asetuksista sekä instanssien lukumäärän voi vaihtaa myös hallintaportaalista Scale-välilehdeltä (kuva 20).

4.1.3 Hinnoittelu

Azuren käyttö on maksullista. Microsoft on tosin pitkän aikaa tarjonnut sivuillaan kolmen kuukauden testijaksoa ilmaiseksi. Testijaksolla saa käyttöön lähes kaikki ominaisuude ja jakson jälkeen kehittäjällä on varmasti hyvä kuva siitä, mistä palveluista haluaa tämän jälkeen maksaa. Testijakson käyttöönottoon tarvitsee Microsoft-tilin (entinen Live-tili) sekä luottokortin. Vaikka luottokorttitiedot pitää syöttää, ei palvelu maksa mitään, vaan tilille on määritetty nollan euron kulutuskatto. Luottokorttia käytetään henkilöllisyyden varmistamiseen ongelma- tai väärinkäyttötilanteissa.

Azuressa asiakas maksaa käytetyistä tehoista, liikenteestä ja esimerkiksi tietokannan hinnoittelu menee tietokannan koon mukaan. Cloud Servicessä asiakas maksaa mm. instanssien lukumäärän ja virtuaalikoneen koon (tehojen) perusteella. Liikenne Azuren suuntaan on ilmaista, mutta liikenne Azuresta ulospäin on maksullista.

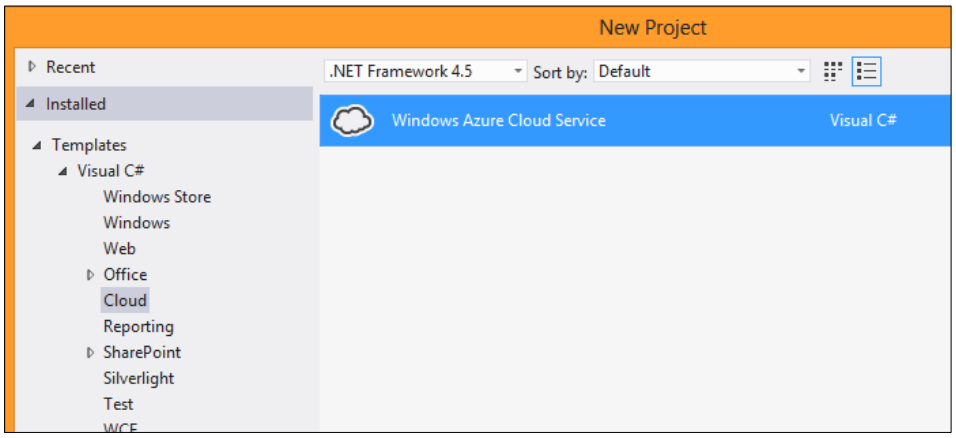
Azure-palvelulla on olemassa erilaisia hinnoitteluohjelmia, joilla voidaan määrittää ja ennakoida tulevia laskuja. ”Pay-As-You-Go” on käyttöperustainen hinnoittelu, kun taas 12-month tai 6-month -maksuohjelmat mahdollistavat muun muassa maksun etukäteen sekä erilaiset paljousalennukset ja ennakoidut kuukausimaksut. (Windows Azure Pricing Calculator 2013.)

Pay-As-You-Go -hinnoittelusta puuttuu kuitenkin yksi ikävä ominaisuus. Se on käyttörajoitus, jolla voitaisiin määrittää tietty maksimi summa kuu-

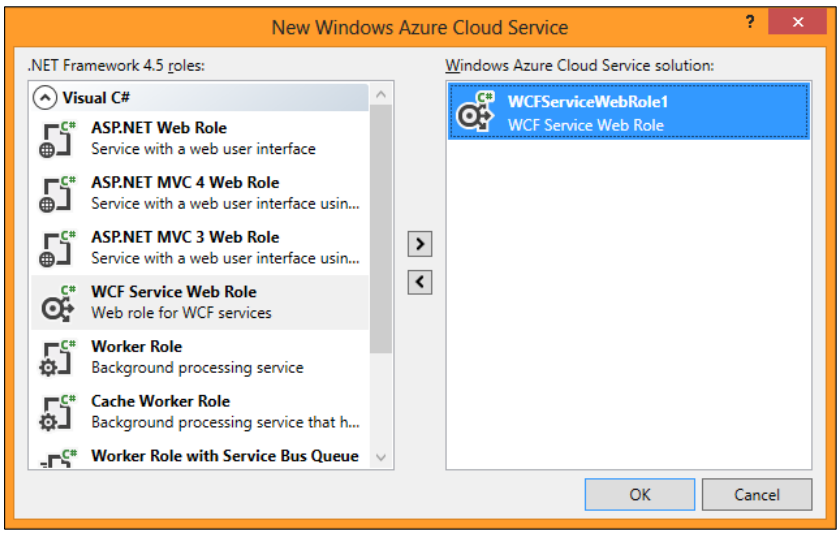
kausittaiselle laskulle, esimerkiksi kymmenen euroa. Kuukausittaisen laskun yltäessä kymmeneen euroon palvelut voitaisiin ajaa alas, eikä ylimääräistä laskua enää syntyisi.

4.2 Käyttöönotto ja asetukset

Azure on helppo ottaa käyttöön. Peruskäytössä asetuksiin tarvitsee koskea todella vähän ja portaalin käytön oppii hetkessä. Visual Studiossa projektipohjista löytyy Cloud-kohdasta yksi vaihtoehto, Windows Azure Cloud Service (kuva 12). Azure Cloud Serviceen voidaan lisätä joko Web Roleja tai Worker Roleja (kuva 13). Web Rolet ovat joko, ASP.NET-verkkopalveluita tai WCF-palveluita. Worker Roleja käytetään taustalla ajettaviin prosesseihin. Jotta Visual Studiolla voi alkaa luoda Azure-pohjaisia toteutuksia, pitää kehittäjän ladata Microsoftin sivuilta Azure SDK.



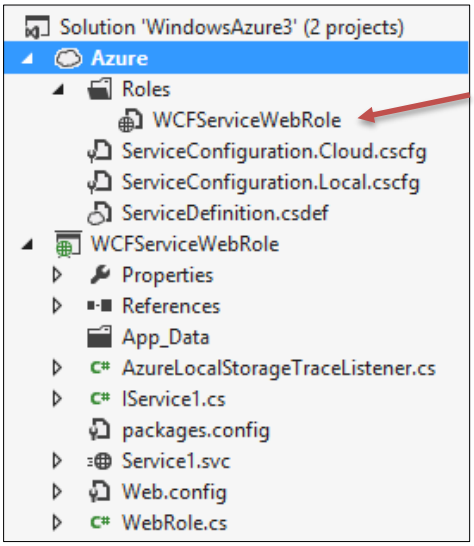
Kuva 12. Azurelle löytyy vain yksi projektipohja



Kuva 13. Cloud Servicelle saa monenlaisia rooleja

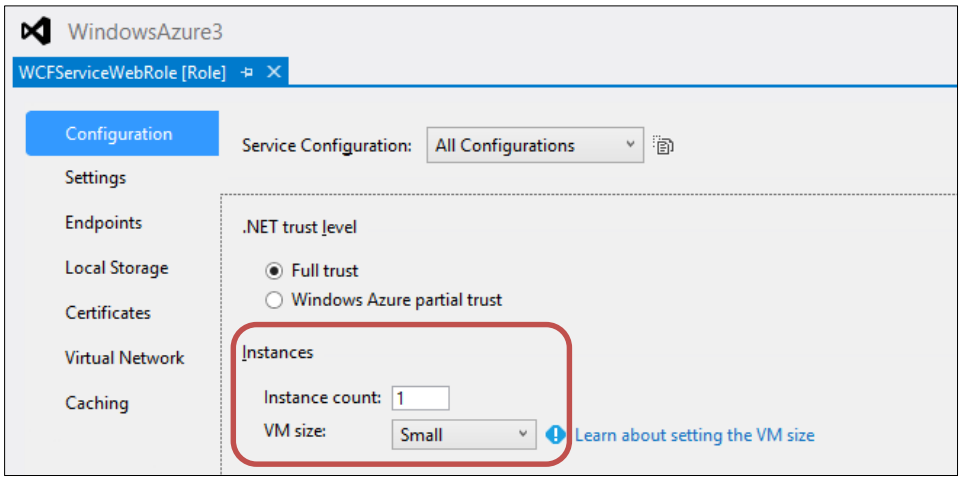
Cloud Service -projektipohja sisältää ainakin kaksi projektia, toinen on Azure Cloud Service -projekti ja toinen on projektipohja valitulle roolille,

tässä tapauksessa WCF-projekti (kuva 14). Myöhemmin voidaan lisätä esimerkiksi toinen WCF-projekti, mikäli tarvitaan kahta erillistä palvelua.



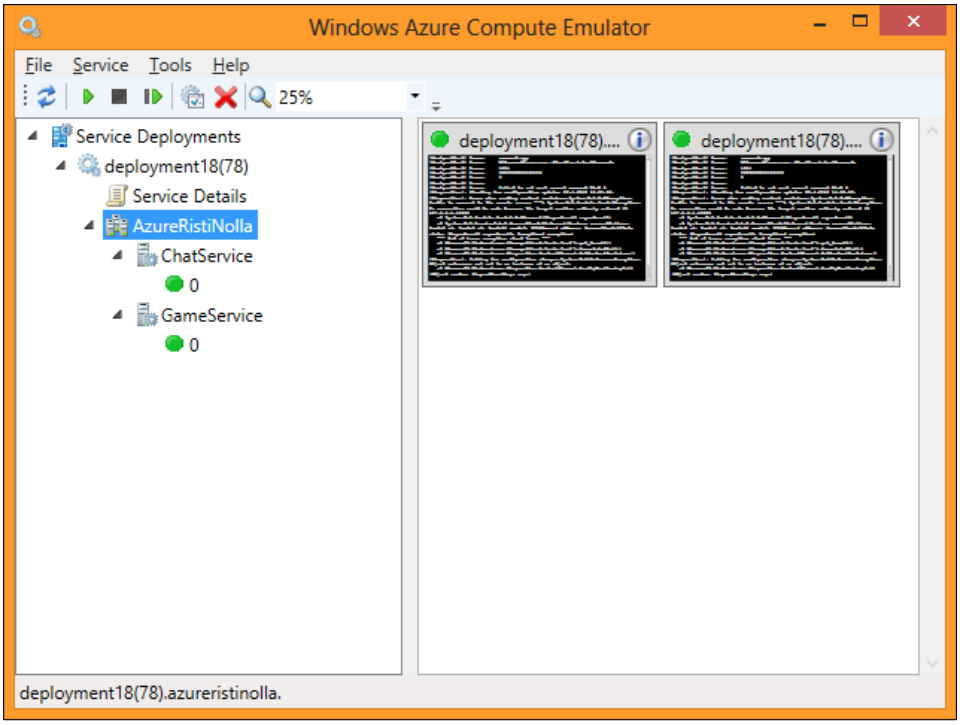
Kuva 14. Tyhjän projektin generoidut tiedostot yhdellä roolilla

Azure Cloud Service -projektin rooli(e)n asetuksiin (kuva 15.) voidaan määritellä käytetyn virtuaalikoneen, eli koneen, jolla palvelu tullaan isännöimään, koko. Instanssien lukumäärä voidaan määrittää myös roolin asetuksista (kuva 15.).



Kuva 15. Roolin asetuksiin pääsee käsiksi Visual Studiossa, asetukset eivät tule tietenkään voimaan ennen, kuin uusi versio on ajettu Azurelle. Instanssien määrää voidaan korottaa myös Azuren hallintaportaalissa, jolloin muutos tapahtuu heti.

Projektia voidaan ajaa sekä debugata paikallisesti esimerkiksi IIS Express -palvelimella samaan tapaan, kuin mitä tahansa WCF-projektia. Azure SDK:n asennuksessa Visual Studio saa Azure Compute - sekä Storage emulaattorin. Emulaattoreilla voidaan esimerkiksi tarkastella jokaista roolia ja niiden instansseja (kuva 16).

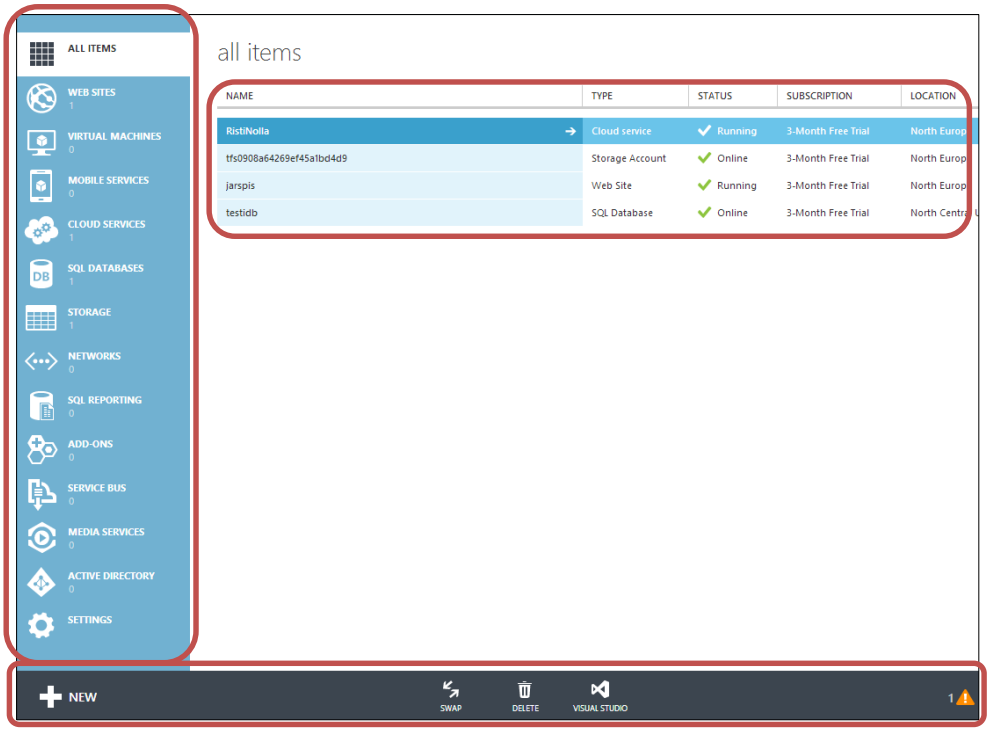


Kuva 16. Compute Emulatorissa instansseja eli virtuaalikoneita voidaan tarkastella komentorivitasolla

4.3 Hallintaportaali

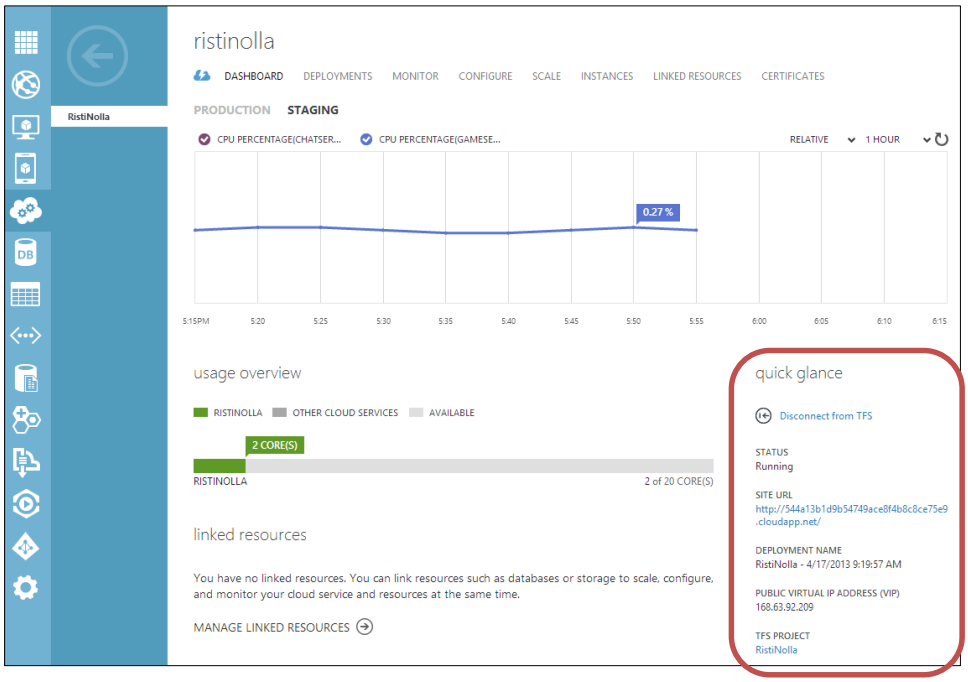
Azuren hallintaportaalia käytetään selaimen avulla ja se on saatu hyvin yksinkertaiseksi ja eri toiminnot löytyvät sieltä helposti. Portaalista voidaan lisätä ja konfiguroida käytössä olevia palveluita. Azuren Account-sivustolla voidaan tarkastella oman tilauksen tilaa, muuttaa maksusuunnitelmaa tai yhteystietoja. Sivustolta löytyvät vanhat laskut sekä järjestelmä osaa ennakoida myös tulevaa laskua.

Azuren hallintaportaalin etusivulla (kuva 17) on yhteenveto käytetyistä palveluista, niiden tilasta, tilauksista ja palveluiden sijainnista. Vasen navigointipalkki avaa eri palveluiden tarkemman näkymän. Alapalkissa on toiminto uusien palveluiden lisäämiseksi sekä toiminnot valitulle palvelulle.



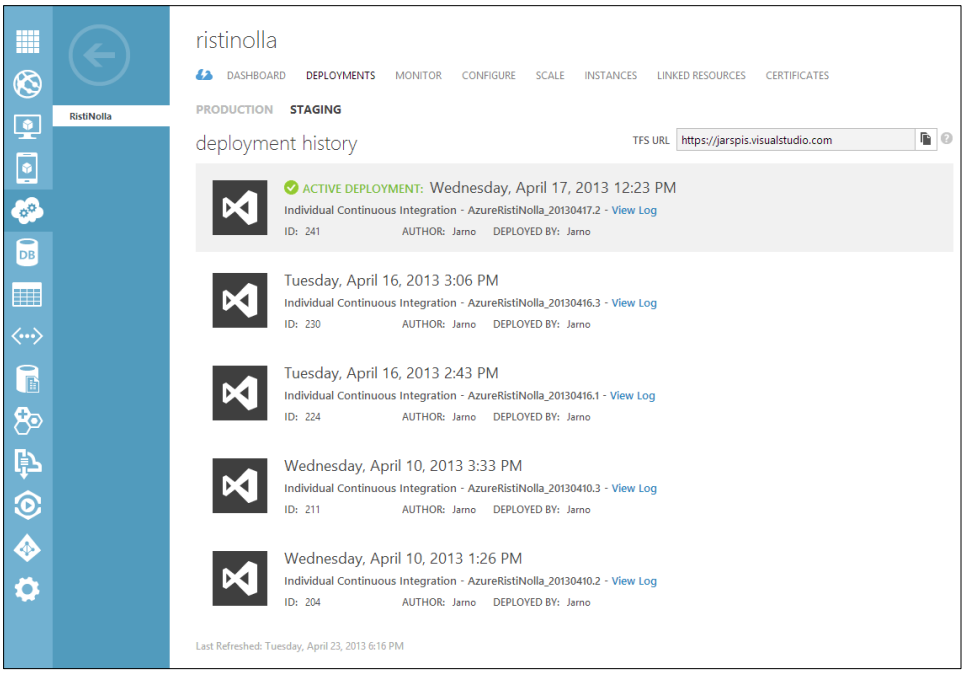
Kuva 17. Hallintaportaalin etusivu

Cloud Servicen etusivulta (kuva 18) oikealta löytyvät palvelun perustiedot osoitteista, tiloista, käyttöhistoriasta sekä instansseista. Cloud Servicen oma navigaatio löytyy ylhäältä vaakarivistä.



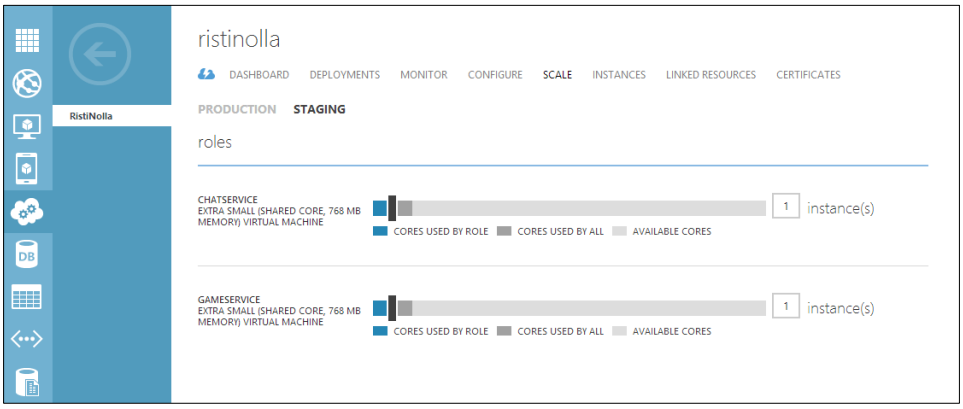
Kuva 18. Cloud Service -etusivu

Deployment-välilehdeltä (kuva 19) löytyy versioiden ajohistoria, tässä tapauksessa Cloud Service oli yhdistetty Microsoftin versiohallintapalvelu Team Foundation Service -projektiin, jonka kautta ajettiin uudet versiot palvelimelle. Lisää Team Foundationin integraatiosta luvussa 4.4..



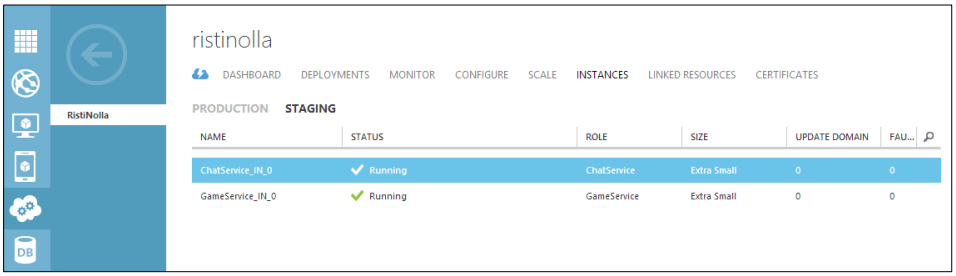
Kuva 19. Deployment-historia

Scale-välilehdeltä (kuva 20) voidaan lisätä instanssien lukumäärää rooleittain. Sama asetus, joka löytyy Visual Studiosta roolin asetuksista (kuva 15). Instanssien lukumäärän kasvattaminen toimii portaalissa välittömästi, uudet instanssit ajetaan päälle parissa minuutissa. Uusi versio pitää ajaa palvelimelle, jos projektin asetuksia muutetaan Visual Studion kautta, jotta ne tulevat voimaan.



Kuva 20. Instansseja voidaan lisätä liukusäätimellä

Instances-välilehdeltä (kuva 21) löytyy roolien instanssien tilat ja muuta tietoa, kuten virtuaalikoneen koko.

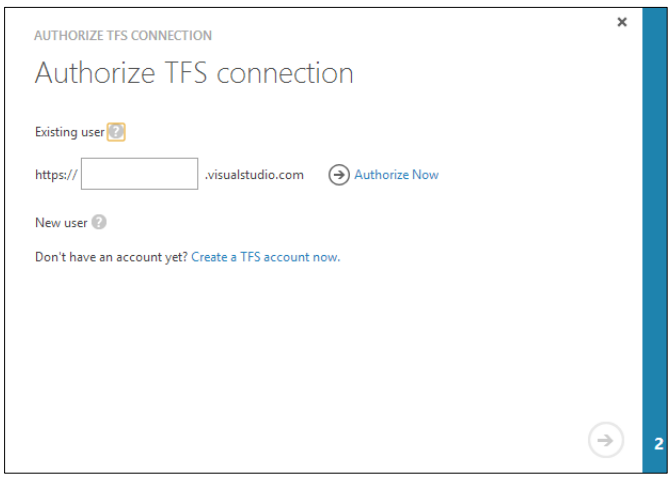


Kuva 21. Intances-välilehti

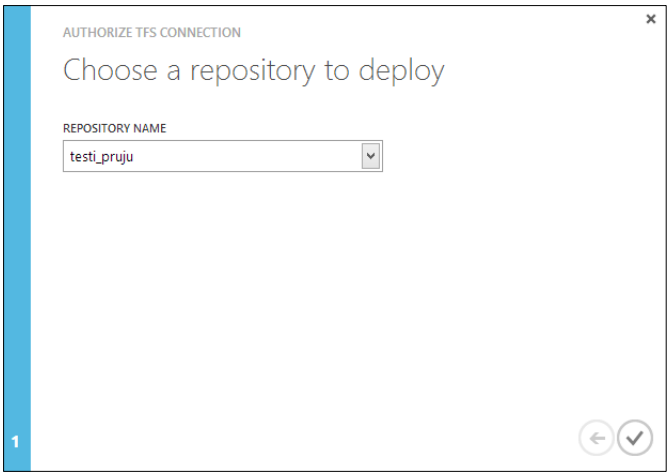
4.4 Team Foundation integraatio

Microsoft tarjoaa kehittäjille ilmaiseksi käyttöön oman versiohallintaohjelmiston pilviversiota, eli Team Foundation Servicen. Sen käyttöönotto on yksinkertaista ja kannattavaa varsinkin hieman suuremmissa kehitystöissä. Ohjelmistokehityksen kannalta versiohallinta on hyvä valinta, sillä lähdekoodit ovat aina saatavilla paikasta ja ajasta riippumatta. Koodien yhdistäminen on helppoa myös silloin, kun kehitystyötä tekee useampi henkilö. Versiohallinta tarjoaa lähdekoodille historian ja vertailun eri versioiden välillä. Ilmaisessa versiossa Team Foundation -projektin kehitystiimin maksimikoko on viisi henkilöä.

Team Foundationin avulla uusien versioiden saaminen palvelimelle on tehty helpoksi. Kehittäjän täytyy yhdistää Team Foundation Servicen tiimiprojekti Azuren Cloud Serviceen (kuvat 22, 23 ja 24) ja määrittää TFS-tiimiprojektin Build-asetuksiin yhteys Azureen (Individual Continuous Integration build). Tällöin jokaisen uuden koodin palvelimelle lähetyksen jälkeen (Check in), TFS koostaa koodin, ja Azure saa TFS-palvelulta uuden palveluversion ja ajaa uuden version käyttöön.

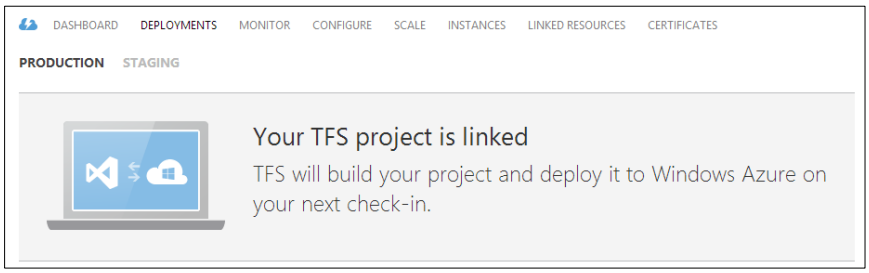


Kuva 22. TFS-määrityksessä syötetään oman TFS-tilin osoite



Kuva 23. Kun TFS-tili löytyy, valitaan oikea Team Project

Team Foundationin lähdekoodin koostamisessa ja Azuren uuden version käyttöönotossa (deployment) kestää noin viisi minuuttia. Ominaisuuden saa myös pois päältä muuttamalla samoja Build-asetuksia, jolloin jokaisen pienen muutoksen takia ei tarvitse päivittää palvelimella ajettua versiota.



Kuva 24. Deployment-välilehdeltä huomaa, että TFS-projekti on liitetty onnistuneesti

Kattavat ohjeet Team Foundation Servicen integraatioon löytyvät Micosoftin Windows Azure -sivustolta (Continous delivery to Windows Azure by using Team Foundation Service 2012.). Ohje sisältää myös Team Foundation Servicen käyttöönoton ja siinä on eritelty kattavasti tarvittavat eri asetukset ja määritykset, niin Azuren, kuin Visual Studion puolella.

5 KÄYTÄNTÖ

Käytännön osuudessa oli päämääränä kehittää verkkototeutus Ristinolla-pelin moninpeliä sekä chat-keskustelua varten. Tarvittiin kaksi toteutusta, toinen Windows Phone 8 -alustalle sekä toinen Windows Store 8 -alustalle. Moninpelin oli tarkoitus toimia alustariippumattomana, jolloin esimerkiksi PC:llä voisi pelata matkapuhelimen käyttäjää vastaan. Verkkototeutus tehtiin Windows Communication Foundationin sekä Azuren Cloud Servicen avulla.

5.1 Palveluiden esittely

Palvelukokonaisuus pyörii Azure Cloud Servicenä jaetulla pienimmällä mahdollisella virtuaalikoneella (xs, eli extra small). Cloud Servicellä on kaksi WCF-palvelua. Ensimmäinen WCF-palvelu on ChatService ja toinen on GameService. Molemmat palvelut ovat arkkitehtuuriltaan ja suunnittelultaan hyvin samankaltaisia. Ne käyttävät Callback-sopimuksia ja molemmilla on kaksi päätepistettä, toinen Windows Phone 8 ja toinen Windows Store 8 -sovellusta varten.

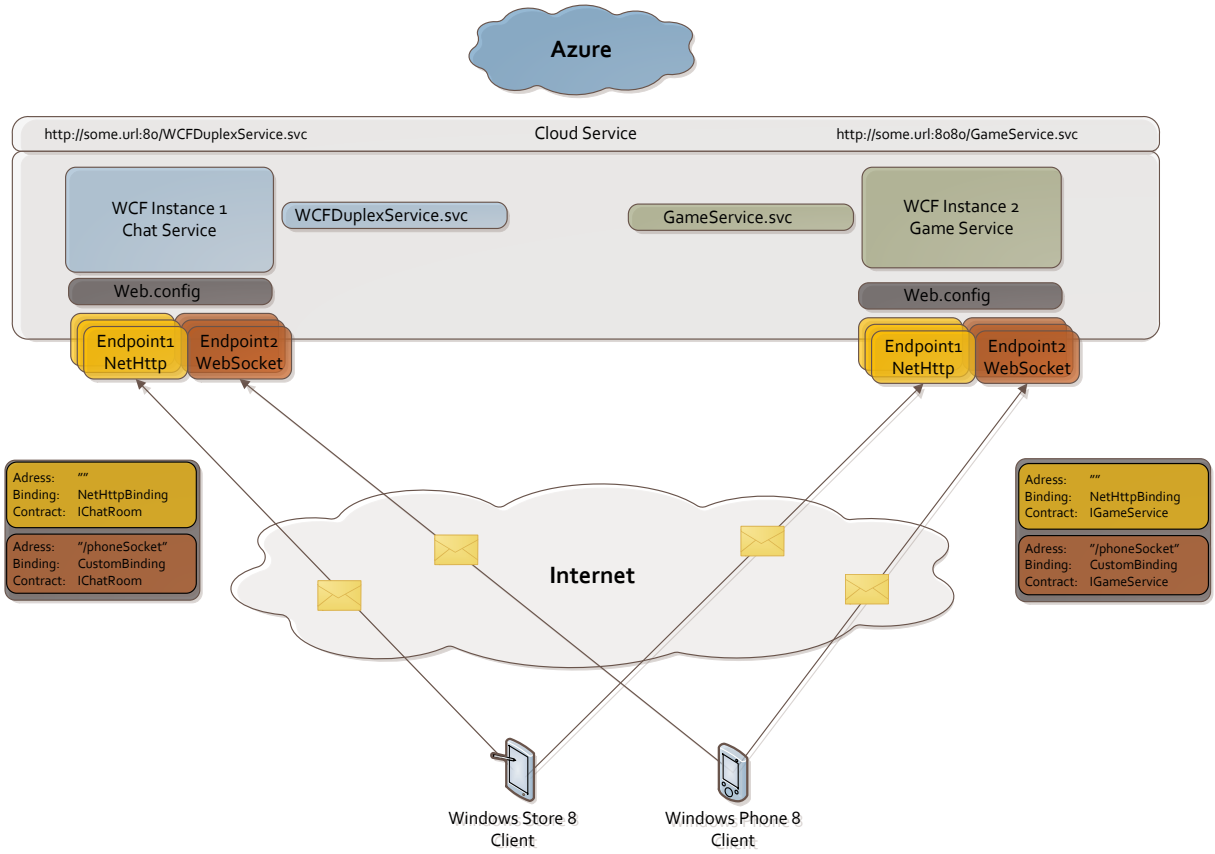
ChatService-palvelu välittää käyttäjien lähettämät viestit palvelimen kautta käyttäjille. Chat-keskustelu toimiin ikään kuin pelin aulana, jolloin online-käyttäjät voivat keskustella keskenään. GameService välittää ChatServicelle viestien avulla uudet pelit sekä peliin liittymiset. Käyttäjän tehdessä pelin chat-keskusteluun tulee pelin tekijältä viesti, jossa kerrotaan, että kyseinen käyttäjä on luonut pelin, kooltaan x*x ja hän odottaa vastustajaa. Kun vastusta liittyy peliin, GameService välittää siitä tiedon chatille ja käyttäjältä tulee automaattisesti viesti peliin liittymisestä. Tällöin online-käyttäjät näkevät uudet pelit ja niihin liittyvät pelaajat. ChatServicen lähdekoodi on liitteessä 1.

GameService välittää pelaajien siirrot pelilaudalla ja huolehtii vastustajien löytämisestä sekä käynnissä olevien pelien kirjanpidosta. Käyttäjä voi luoda pelin haluamallaan ruutukoolla ja myös peliin liittyvä käyttäjä saa etsiä pelejä ruudukon koon mukaan. GameService lähdekoodi on liitteessä 2.

Molemmat palvelut käyttävät samaa duplex-viestinvälitysmallia (MEP). Palvelut olisi todennäköisesti voinut myös tehdä yhdeksi samaksi palveluksi (yksi WCF-projekti), mutta suunnitteluvaiheessa vaikutti parhaimmalta ratkaisulta tehdä palvelut omiksi projekteiksi. Tällöin saadaan palveluista vikasietoisemmat – jos toinen palveluista kaatuu, ei se vaikuta toisen toimintaan millään tapaa. Myös Azuren suorituskykysäädöt ja instanssien määrää voidaan muuttaa molemmille sopiviksi tarpeen mukaan.

WCF-palvelut on toteutettu ilman tietokantaa, sillä molemmat palvelut käyttävät staattisia listoja, joilla pidetään kirjaa online-käyttäjistä sekä tallennetaan tiedonsiirtokanava palvelimen ja käyttäjän välille. Käynnissä olevien pelien ylläpito hoidetaan niin ikään staattisia listoja hyväksi käyttäen.

Kuvasta 25 selviää Azure Cloud Servicen kokonaisuus. Cloud Servicessä on kaksi Web Rolea, eli WCF-palvelua. Molemmille on määritelty kaksi päätepistettä. Cloud Service -laatikon sisäpuolella on kaksi WCF-palvelua. Laatikoiden keskellä ovat *.svc-tiedostot, jotka ovat rajapintojen toteuttavat luokat ja sisältävät WCF-palveluiden käyttämän liiketoimintalogiikan. Web.config-tiedostoihin on määritetty esimerkiksi päätepisteet ja sidokset. Cloud Servicen alimmat osat eli keltaiset ja oranssit muodot tarkoittavat päätepisteitä (Endpoint1 ja Endpoint2) ja niiden kolme kerrosta, päätepisteiden perusrakennetta (ABC). Päätepisteiden rakenne on selvennetty samoilla väreillä alempana. Palveluiden asiakkaina toimivat kuvan alareunassa Windows Store 8 ja Windows Phone 8 -sovellukset. Molemmille laitteille on palvelussa omat päätepisteensä.



Kuva 25. Palvelinsovellusten arkkitehtuuri

5.2 Palveluiden suunnittelu

Palveluiden suunnittelussa piti ottaa huomioon eri asiakasohjelmien vaatimukset ja ominaisuudet, sillä samat tekniikat eivät toimineet sekä WS8 että WP8-alustalla, kuten teoriaosassa todettiin. BasicHttpBinding olisi ollut yksinkertainen ja yhteensopiva sidos lähes jokaisella alustalla yksinkertaisuutensa ansiosta, mutta sen tarjoamat ominaisuudet olivat liian suppeat. Se tuki vain pyyntö-vastaus -mallia, joka ei työssä olisi riittänyt, vaan tarvittiin tuki duplex-mallille. Läheskään kaikki WCF:n tarjoamat sidokset eivät myöskään tue duplex-mallista tiedonsiirtoa. Kehitys aloitettiin Windows Store 8 -alustalta ja liikkeelle lähdettiin NetHttpBinding-sidoksella.

5.2.1 Sidokset ja päätepisteet

NetHttpBinding on uuden .NET Framework 4.5:n mukanaan tuoma WCF:n tukema sidos, joka mahdollistaa sekä BasicHttpBinding-tyyppiset ominaisuudet että Duplex-mallisen tiedonsiirron web-socketin avulla. Sidos tekee tiedonsiirron oletuksena SOAP:n sijaan binäärinä ja se on tuettu tällä hetkellä vain Windows 8 -alustalla. (NetHttpBinding Class 2012.)

WP8-alustan tukemat sidokset, eivät suoraan tukeneet duplex-mallia halutulla tavalla, joten työssä päädyttiin käyttämään CodePlex-sivustolla kehitettyä WebSocket4Net-kirjastoa. Kirjasto mahdollistaa http-protokollan yli

duplex-tiedonsiirron web-socketin avulla. WebSocket4Net-kirjasto antaa kehitysmahdollisuudet monille alustoille esimerkiksi .NET (versiot 2.0 - 4.5), Silverlight ja Mono (Android). WebSocket4Net-kirjasto otetaan käyttöön luomalla oma sidos, eli CustomBinding. Esimerkissä 10 on ote Chat-Servicen Web.config-tiedostosta, jossa määritellään oma sidos ja sitä käyttävä päätepiste.

```
<customBinding>
   <binding name="webSocket">
      <byteStreamMessageEncoding/>
      <httpTransport>
         <webSocketSettings transportUsage="Always"
             createNotificationOnConnection="true"/>
      </httpTransport>
   </binding>
</customBinding>

<endpoint address="phoneSocket"
          binding="customBinding"
          bindingConfiguration="webSocket"
          contract="ChatService.IChatRoom"
/>
```

Esimerkki 10. CustomBinding ja sidoksen käyttö päätepisteessä

5.2.2 Callback Contract

WCF-palveluissa käytetty tiedonsiirtomalli oli duplex, jolloin WCF-palveluille pitää määrittää Callback Contract. Callback Contract määrittää WCF-palvelulle rajapinnan serveriltä asiakasohjelmaan päin. Koodiesimerkissä 11 määritetään CallbackContract-attribuutilla serverin käyttämä rajapinta asiakkaan suuntaan. Asiakassovellus toteuttaa Callback-sopimuksessa määritetyn rajapinnan.

Esimerkissä 11, IChatRoom on rajapinta palvelimella, jonka palvelin toteuttaa (.svc -tiedostossa) ja jota asiakasohjelma käyttää yhteydessä WCF-palveluun. Rajapinnassa kaikki metodit on merkitty IsOneWay-attribuutilla, joka määrittää metodin yksisuuntaiseksi. Nyt metodin täytyy olla siis void-tyyppinen eli sillä ei ole paluuarvoa. Esimerkin viimeinen metodi, UnidentifiedMessage, on tarkoitettu WP8-alustalle, sillä WP8-sovellus ei kutsu muita rajapinnan metodeja. WP8-sovellus lähettää Message-luokan viestin rajapinnalle, jonka sisällöstä parsitaan viesti ja switch-case -rakenteessa kutsutaan haluttu toiminto (liite 1 - WCF ChatService).

```
[ServiceContract(
   Name = "ChatRoom",
   SessionMode = SessionMode.Required,
   CallbackContract = typeof(IChatRoomCallback))]

public interface IChatRoom
{
  [OperationContract(IsOneWay = true)]
  void Subscribe(string clientName, bool isWebSocketClient = false);

  [OperationContract(IsOneWay = true)]
  void Unsubscribe(string clientName);

  [OperationContract(IsOneWay = true)]
  void SendBroadcastMessage(string clientName, string message);

  [OperationContract(IsOneWay = true)]
  void SendPrivateMessage(string clientName, string message,
                          string recipientName);

  [OperationContract(IsOneWay = true, Action = "*")]
  void UnidentifiedMessage(Message message);

}
```

Esimerkki 11. IChatRoom-rajapinta, rajapinnan toteuttava luokka liitteessä 1

Esimerkissä 12, IChatRoomCallback on palvelimella sijaitseva rajapinta, jonka asiakassovellus toteuttaa ja jota WCF-palvelu käyttää ollessaan yhteydessä asiakkaaseen. WCF-palvelu kutsuu rajapinnan metodeja ja asiakassovelluksen huoleksi jää liiketoimintalogiikan toteuttaminen. IChatRoomCallback on määritelty IChatRoom-rajapinnan CallbackContract:ssa. GameService toteuttaa samantyyppisen Callback-sopimuksen kuin ChatService.

```
public interface IChatRoomCallback
{
  [OperationContract(IsOneWay = true)]
  void NotifyClientJoined(string clientName);

  [OperationContract(IsOneWay = true)]
  void NotifyClientLeft(string clientName);

  [OperationContract(IsOneWay = true)]
  void NotifyMessage(string message);

  [OperationContract(IsOneWay = true)]
  void NotifyUsernameChanged(string clientName);

  [OperationContract(IsOneWay = true, Action = "*")]
  void SendMessage(Message message);
}
```

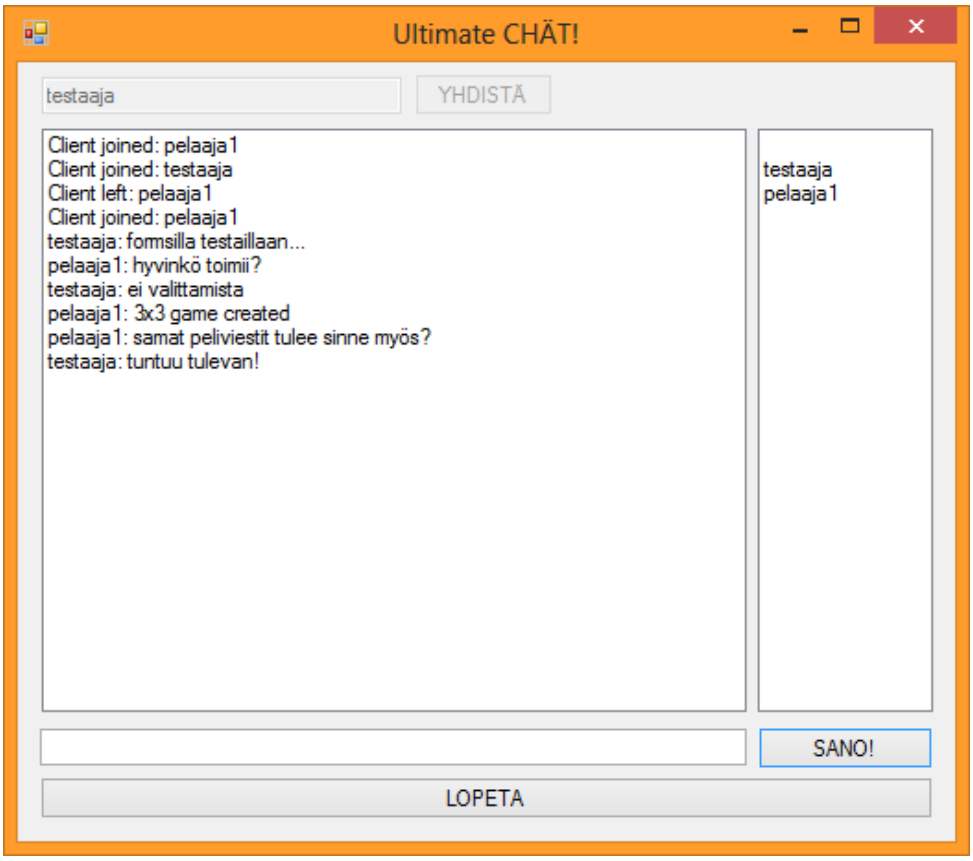
Esimerkki 12. IChatRoomCallback

5.3 Esimerkit palvelun käytöstä

Kehityksen alkuvaiheessa tekniikoita kartoittaessa löydettiin tarkoituksiin sopivan oloinen sidos, eli jo läpi käyty NetHttpBinding. Ohjelmistokehitys

aloitettiin Chat-sovelluksesta, jotta voitaisiin todeta sidoksen sopivuus tarkoitukseen sekä sen suurimmat ongelmat. Chat-sovelluksen testausta varten tehtiin yksinkertainen Windows Forms -käyttöliittymän omaava testiohjelma (kuva 26). Windows Formsilla tehty sovellus sisälsi kaikki samat ominaisuudet, jotka otettiin käyttöön sekä Store 8- että Phone 8 -alustoilla. Formseilla testaamisen jälkeen NetHttpBinding havaittiin erittäin sopivaksi käyttötarkoitukseen. NetHttpBinding-tuen puuttuminen WP8-alustalta, pakotti WebSocket4Net-kirjaston testaamisen samaisella Windows Forms -testisovelluksella. Testaamisen jälkeen WebSocket4Net-kirjasto havaittiin niin ikään sopivaksi WP8-alustalle.

Onnistuneiden testien jälkeen palvelun toisen osan eli GameServicen toteutus päätettiin tehdä pääpiirteittäin samalla kaavalla käyttäen WS8-alustalla NetHttpBindingiä ja WP8-alustalla WebSocket4Net-kirjastoa. Myös GameService tarvitsi molemmille asiakassovelluksille omat päätepisteensä. GameServicelle ei sen suurempaa testausta järjestetty, lukuun ottamatta testausta muutaman rivin konsolisovelluksella, jolla palvelu todettiin toimivaksi.



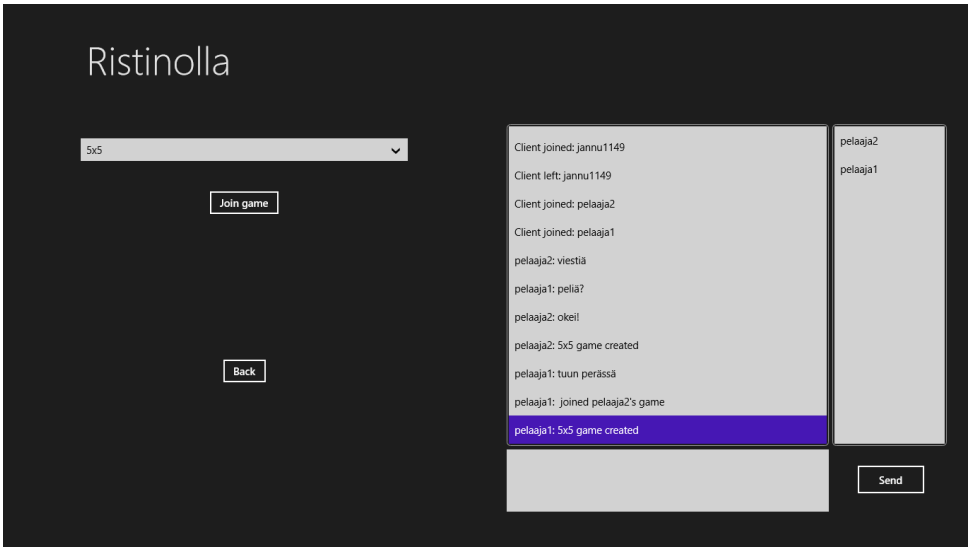
Kuva 26. Windows Forms -toteutus WCF-Chat -rajapinnasta

WCF-palvelun kutsuminen WS8-alustalla eroaa hieman esimerkiksi konsolisovelluksesta. Viittaus palveluun (Service Reference) tapahtuu samaan tapaan, mutta viittauksesta ei generoida konfiguraatiotiedostoa (app.config), sillä konfigurointi tapahtuu lähdekoodissa. Viittaus palveluun generoidaan References.cs-tiedostoon. Kaikki WCF-palvelun toiminnot Windows Store alustalla ovat asynkronisia. Palvelun toimintojen kutsuminen tapahtuu esimerkin 13 mukaan. (Accessing WCF Services with a Windows Store Client App 2012)

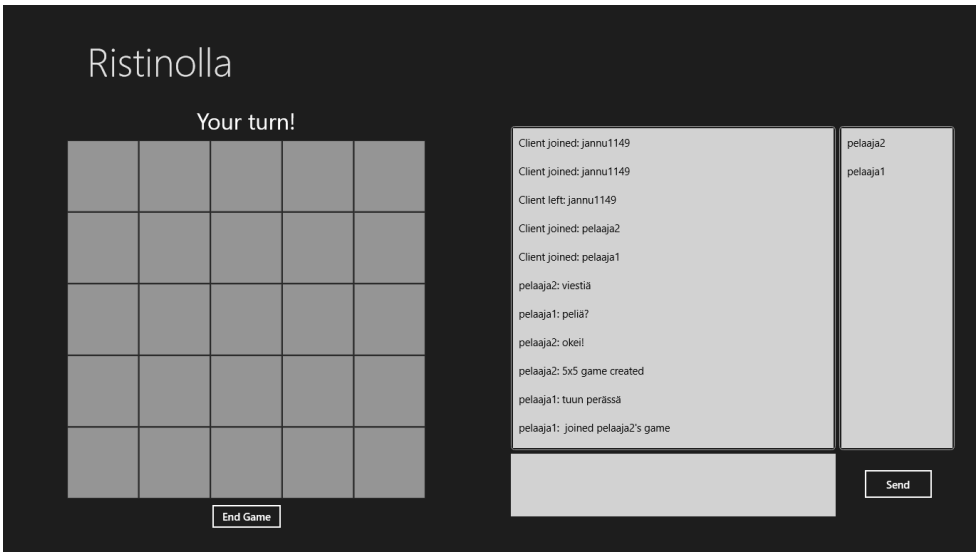
```
void async JokuMetodi()
{
    ServiceClient client = new ServiceClient();
    Task<T> results = await client.GetDataAsync(66);
    T result = results.Result;
    if (result.Success)
    {
       // Tänne WCF-palvelun palauttaman tuloksen käsittely
    }
}
```

Esimerkki 13. WCF-palvelun kutsuminen Windows Store 8 -alustalla

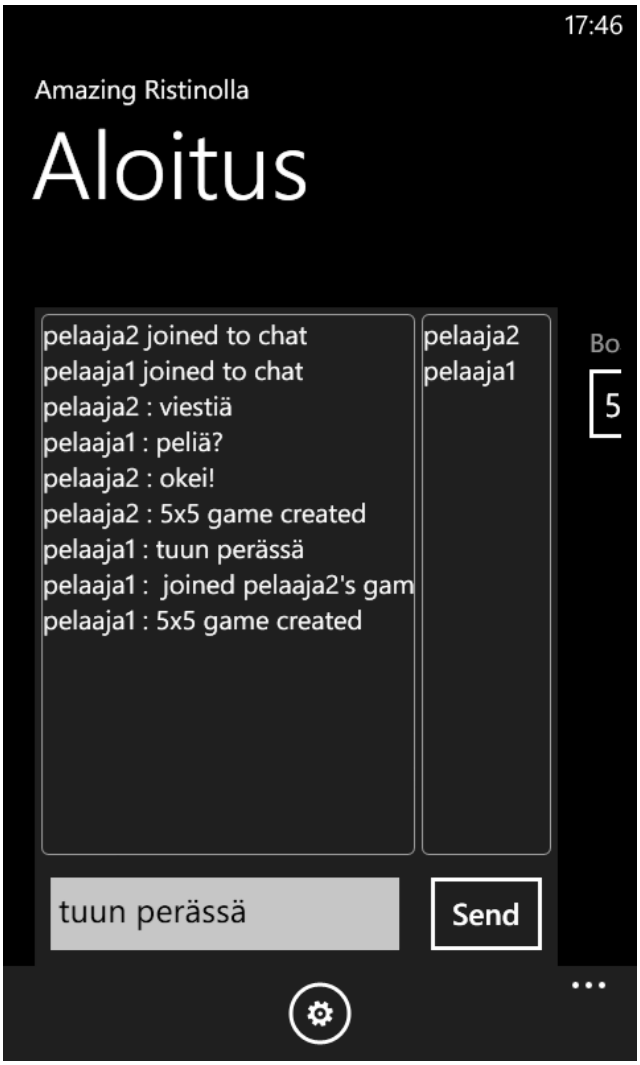
Kuvat 27 ja 28 esittävät Windows Store 8 -alustan Ristinolla-pelin työn aikaista versiota käyttöliittymästä. Käyttöliittymä tehtiin aikataulun sanelemana hyvinkin pelkistetyksi. Kuvat 29 ja 30 ovat Windows Phone 8 -sovelluksesta.



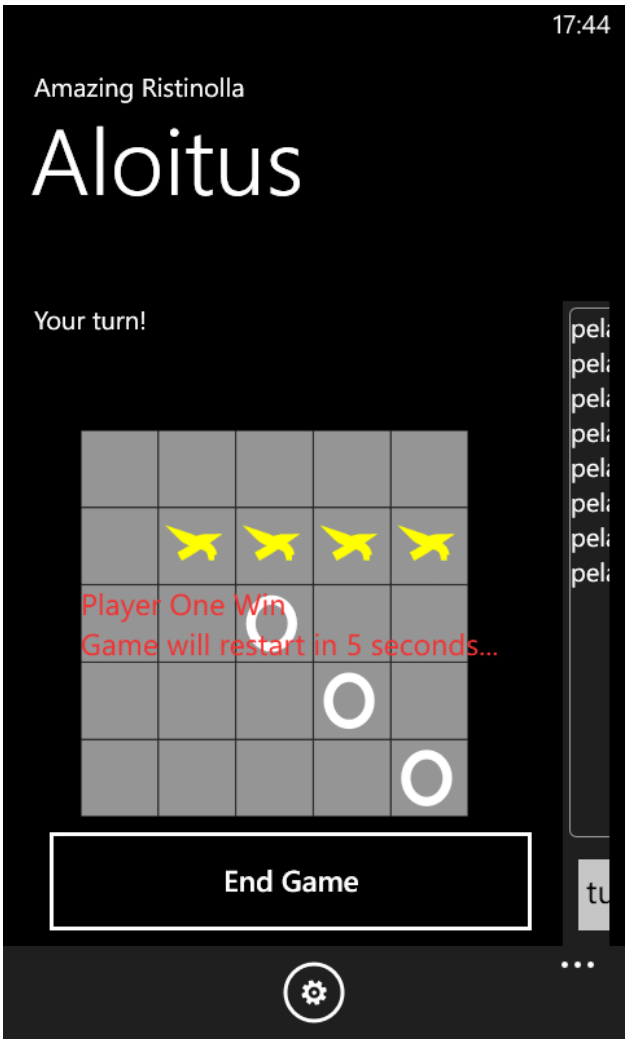
Kuva 27. Windows Store 8 -sovelluksen etusivu sisältää chatin sekä valikon pelin luomiseksi tai peliin liittymiseksi



Kuva 28. Näkymä pelin ollessa käynnissä



Kuva 29. Windows Phone 8 -käyttöliittymä käyttää panorama-sivuja



Kuva 30. Windows Phone 8 -käyttöliittymän pelilauta

5.4 Jatkokehitys

WCF-palvelusta tehtiin kevyt versio, sillä se ei käytä ollenkaan tietokantaa pelien tai chat-keskustelun ylläpitämiseen. Kaikki toiminnot pyörivät staattisilla listoilla, jotka pitävät huolta yhteydessä olevista asiakkaista. Jos palvelua laajentaisi sisältäämään esimerkiksi ranking-listoja tai muita tilastoja, tarvittaisiin palvelun vierelle myös tietokanta. Tietokannan voisi toteuttaa esimerkiksi tavallisella MS SQL -tietokannalla tai ottamalla käyttöön Azuren Mobile Servicen, jossa on tuki yksinkertaisille tauluille. WCF-palvelun tietoturva on konfiguroimatta, joilloin palvelua käyttävää asiakassovellusta ei millään tavalla todenneta tai varmisteta, joten ulkopuolinen palvelun käyttäjä voisi välittää vääränlaisia viestejä muille palvelun käyttäjille tai palvelulle, jolloin palvelun tai asiakassovellusten liiketoimintalogiikka ei käyttäytyisi halutulla tavalla.

Windows Store 8 -sovelluksen käyttämä sidos, eli NetHttpBinding tuntui olevan hyvinkin vakaa verrattuna Windows Phone 8 -sovelluksen käyttämään WebSocket4Net-kirjastoon. WebSocket:lla saatiin muutamia erikoisia virheitä ja toimimattomuuksia. Palvelun virheiden etsintä hankaloituu, kun palveluita ajetaan pilvestä. Silloin ei esimerkiksi päästä seuraamaan

lähdekoodia breakpointeilla. Paikallisesti myös samojen virheiden saaminen kuin pilvessä ei ollut aina mahdollista. Nähtäväksi jää, tuovatko päivitykset WP8-alustalla myös tuen tulevaisuudessa NetHttpBindingille. Tällä hetkellä se toimii vain .NET 4.5 versiolla ja Windows 8 -alustalla.

Suurempia suorituskykytestejä ei palvelulle ehditty tekemään, ainoastaan muutamien samanaikaisten käyttäjien kanssa. Azuren xs-kokoisella virtuaalikoneella palvelut kuitenkin tuntuivat toimivan ilman viivettä, eikä Azuren hallintaportaalin tilastoista suuria prosessorikäytön vaihteluita löydetty. Näiden tietojen perusteella pidimme palvelua kohtuullisen kevyenä.

6 YHTEENVETO JA PÄÄTELMÄT

Tekijällä oli alun perin hieman kokemusta Windows Communication Foundationin käytöstä. Käyttökokemusta oli tullut perusteista käyttäen sidoksena BasicHttpBindingiä, jolloin palveluiden toimintaan saattamiseksi ei juuri konfigurointia tarvita. Kahden päätepisteen käyttö tuli myös uutena asiana työn edetessä.

Duplex- ja soketti-verkkotekniikat olivat tuttuja ennen työtä vain ajatuksen tasolla. Alun perin palveluita suunniteltiin käytettäväksi tietokantapohjaisena ja yksinkertaisempana, mutta suorituskykyä ja verkkoja pohdiskellessa päädyttiin ottamaan duplex ja soketti mukaan projektiin.

Yllättäviä ongelmia tuotti oikeanlaisten tekniikoiden löytäminen. Asiakassovellusten yhteensopimattomuus sidosten osalta yllätti. WebSocket4Net-kirjaston käyttö ei ollut yhtä selkeää ja yksinkertaista kuin NetHttpBinding-sidoksen kanssa. .NET:n versio 4.5 oli myös kohtuullisen uusi, eivätkä eri kirjat sekä MSDN:n ohjeet aina olleet ajan tasalla uusista toiminnoista ja tekniikoista.

Azure oli alustana alun perin tuiki tuntematon, mutta MSDN-tutoriaalien perusteella Cloud Servicen käyttöönotto oli yllättävän helppoa, kun ei ollut juuri mitään konfiguroitavaa. Azuren käyttö tuli ajankohtaiseksi mietittäessä, miten palvelu tultaisiin isännöimään. Pilvipalvelu oli paras ratkaisu, sillä palvelinkoneita ei ollut omasta takaa heti saatavilla, eikä fyysisten tai virtuaalisten koneiden asentamiseen haluttu käyttää ylimääräistä aikaa.

LÄHTEET

Accessing WCF Services with a Windows Store Client App. 2012. Windows Developer Network. Viitattu 20.5.2013.
http://msdn.microsoft.com/en-us/library/hh556233.aspx

Configuring System-Provided Bindings. 2012. Microsoft Developer Network. Viitattu 12.4.2013.
http://msdn.microsoft.com/en-us/library/ms731092.aspx

Continous delivery to Windows Azure by using Team Foundation Service. 2012. Windows Azure. Viitattu 18.4.2013.
http://www.windowsazure.com/en-us/develop/net/common-tasks/publishing-with-tfs/.

DotNet WebSocket Programming. 2012. CodeProject. Ganesan Senthilvel. Viitattu 8.5.2013.
http://www.codeproject.com/Articles/437342/DotNet-WebSocket-Programming

How to: Use the ChannelFactory. 2012. Microsoft Developer Network. Viitattu 10.5.2013.
http://msdn.microsoft.com/en-us/library/ms734681.aspx

Introduction to Web Services. W3schools. Viitattu 15.5.2013.
http://www.w3schools.com/webservices/ws_intro.asp

Introduction to WSDL. W3schools. Viitattu 15.5.2013.
http://www.w3schools.com/wsdl/wsdl_intro.asp

Introduction to XML. W3schools. Viitattu 15.5.2013.
http://www.w3schools.com/xml/xml_whatis.asp

NetHttpBinding Class. 2012. Windows Developer Network. Viitattu 23.4.2013. http://msdn.microsoft.com/en-us/library/system.servicemodel.nethttpbinding.aspx

Pathak, N. 2010. Pro WCF 4 Practical Microsoft SOA Implementation SE. New York: Apress.

SOAP Envelope Element. W3schools. Viitattu 15.5.213.
http://www.w3schools.com/soap/soap_envelope.asp

Troelsen, A. 2012. Pro C# 5.0 and the .NET 4.5 Framework 6th Edition. New York: Apress.

WCF Test Client. 2012. Microsoft Developer Network. Viitattu 23.4.2013.
http://msdn.microsoft.com/en-us/library/bb552364.aspx

WebSocket4Net. 2012. CodePlex. Kerry Jiang.
Viitattu 8.5.2013. http://websocket4net.codeplex.com/

Windows Azure Cloud Services Concepts. 2013. Channel 9. Haishi Bai. Viitattu 23.4.2013. http://channel9.msdn.com/Series/Windows-Azure-Cloud-Services-Tutorials/Windows-Azure-Cloud-Services-Concepts-Part-1

Windows Azure Pricing Calculator. 2013. Windows Azure. Viitattu 20.4.2013. http://www.windowsazure.com/en-us/pricing/calculator/

WCF CHATSERVICE Liite 1

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net.WebSockets;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.ServiceModel.Channels;
using System.ServiceModel.Web;
using System.Text;
using System.Web;

namespace ChatService
{
    public enum SocketMessageType
    {
        NotifyClientJoined,
        NotifyClientLeft,
        NotifyMessage,
        NotifyUsernameChanged
    }

    [ServiceBehavior(InstanceContextMode = InstanceContextMode.Single,
    ConcurrencyMode = ConcurrencyMode.Multiple)]
    public class WCFDuplexService : IChatRoom
    {
      private static Dictionary<string, IChatRoomCallback> _netHttpClients = new
                     Dictionary<string, IChatRoomCallback>();
      private static Dictionary<string, IChatRoomCallback> _webSocketClients = new
                      Dictionary<string, IChatRoomCallback>();

      public void Subscribe(string clientName, bool isWebSocketClient = false)
      {
        // Subscribe the client to the chat room
        var callback = OperationContext.Current.GetCallbackChannel<IChatRoomCallback>();

            if (callback == null) return;

            // IF there is already another client with same name
            if (_netHttpClients.ContainsKey(clientName) || _webSocketClients.ContainsKey(clientName))
            {
                clientName = clientName + DateTime.UtcNow.Millisecond;

                if (!isWebSocketClient)
                    callback.NotifyUsernameChanged(clientName);

                else
                    callback.SendMessage(CreateMessage(
                              SocketMessageType.NotifyUsernameChanged.ToString() + ";" + clientName));
            }

            // Get other NetHTTP clients
            foreach (var client in _netHttpClients.ToList())
            {
                if (isWebSocketClient == false)
                {
                    callback.NotifyClientJoined(client.Key);
                }
                else
                {
                    callback.SendMessage(CreateMessage(
                              SocketMessageType.NotifyClientJoined.ToString() + ";" + client.Key));
                }
            }

            // Get other weSocket clients
            foreach (var client in _webSocketClients.ToList())
            {
                if (isWebSocketClient == false)
                {
                    callback.NotifyClientJoined(client.Key);
                }
                else
```

```
            {
                callback.SendMessage(CreateMessage(
                          SocketMessageType.NotifyClientJoined.ToString() + ";" + client.Key));
            }
        }

        lock (_netHttpClients)
        {
            if (isWebSocketClient == false)
                _netHttpClients.Add(clientName, callback);
            else
                _webSocketClients.Add(clientName, callback);
        }

        // Notify NetHttp clients
        foreach (var client in _netHttpClients.ToList())
        {
            try
            {
                client.Value.NotifyClientJoined(clientName);
            }
            catch (Exception e) // CLIENT IS GHOST
            {
                Unsubscribe(client.Key);
            }
        }

        // Notify webSocket clients
        foreach (var webSocketClient in _webSocketClients.ToList())
        {
            try
            {
                webSocketClient.Value.SendMessage(CreateMessage(
                          SocketMessageType.NotifyClientJoined.ToString() + ";" + clientName));
            }
            catch (Exception e) // CLIENT IS GHOST
            {
                Unsubscribe(webSocketClient.Key);
            }
        }
    }

    public void Unsubscribe(string clientName)
    {
        // Unsubscribe the client from the chat room
        var callback = OperationContext.Current.GetCallbackChannel<IChatRoomCallback>();

        if (_netHttpClients.ContainsKey(clientName))
            _netHttpClients.Remove(clientName);

        else if (_webSocketClients.ContainsKey(clientName))
            _webSocketClients.Remove(clientName);

        foreach (var client in _netHttpClients)
        {
            try
            {
                client.Value.NotifyClientLeft(clientName);
            }
            catch (Exception e)
            {
                Unsubscribe(client.Key);
            }
        }

        foreach (var webSocketClient in _webSocketClients)
        {
            try
            {
                webSocketClient.Value.SendMessage(CreateMessage(
                          SocketMessageType.NotifyClientLeft.ToString() + ";" + clientName));
            }
            catch (Exception e)
            {
                Unsubscribe(webSocketClient.Key);
            }
```

```
        }
    }

    public void SendBroadcastMessage(string clientName, string message)
    {
        var callback = OperationContext.Current.GetCallbackChannel<IChatRoomCallback>();

        if (callback == null) return;

        foreach (var client in _netHttpClients)
        {
            try
            {
                client.Value.NotifyMessage(clientName + ": " + message);
            }
            catch (Exception e)
            {
                Unsubscribe(client.Key);
            }
        }

        foreach (var webSocketClients in _webSocketClients)
        {
            try
            {
                webSocketClients.Value.SendMessage(CreateMessage(
                          SocketMessageType.NotifyMessage.ToString()
                          + ";" + clientName + " : " + message));
            }
            catch (Exception e)
            {
                Unsubscribe(webSocketClients.Key);
            }
        }
    }

    public void SendPrivateMessage(string clientName, string message, string sendTo)
    {
        var callback = OperationContext.Current.GetCallbackChannel<IChatRoomCallback>();

        if (callback == null) return;

        foreach (var client in _netHttpClients.Where(client => client.Key.Equals(sendTo)))
        {
            try
            {
                client.Value.NotifyMessage(clientName + ": " + message);
            }
            catch (Exception e)
            {
                Unsubscribe(client.Key);
            }
            return;
        }

        foreach (var client in _webSocketClients.Where(client => client.Key.Equals(sendTo)))
        {
            try
            {
                client.Value.SendMessage(CreateMessage(
                          SocketMessageType.NotifyMessage.ToString()
                           + ";" + "Private FROM ---- " + clientName + " : " + message));
            }
            catch (Exception e)
            {
                Unsubscribe(client.Key);
            }
            return;
        }
    }

    public void UnidentifiedMessage(Message message)
    {
        if (message == null)
        {
            throw new ArgumentNullException("message");
```

```
            }

            var property = (WebSocketMessageProperty)message.Properties["WebSocketMessageProperty"];
            var context = property.WebSocketContext;
            var queryParameters = HttpUtility.ParseQueryString(context.RequestUri.Query);
            string content = string.Empty;

            if (!message.IsEmpty)
            {
                byte[] body = message.GetBody<byte[]>();
                content = Encoding.UTF8.GetString(body);
            }

            string str = null;
            if (string.IsNullOrEmpty(content))
            {
                str = queryParameters["Name"].ToString();

                Subscribe(str, true);
            }
            else
            {
                //Sender;Receiver;Message
                string[] splittedContent = content.Split(';');

                if (splittedContent.Length == 3)
                {
                    //Priva
                    SendPrivateMessage(splittedContent[0], splittedContent[1], splittedContent[2]);
                }
                else if (splittedContent.Length == 1)
                {
                    string sender = splittedContent[0];
                    Unsubscribe(sender);
                }
                else
                {
                    string sender = splittedContent[0];
                    str = splittedContent[1];
                    SendBroadcastMessage(sender, str);
                }
            }
        }

        private Message CreateMessage(string content)
        {
            var message = ByteStreamMessage.CreateMessage(
               new ArraySegment<byte>(Encoding.UTF8.GetBytes(content)));

            message.Properties["WebSocketMessageProperty"] = new WebSocketMessageProperty {
               MessageType = WebSocketMessageType.Text };

            return message;
        }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net.WebSockets;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.ServiceModel.Channels;
using System.ServiceModel.Web;
using System.Text;
using System.Web;

namespace GameService
{
    public enum SocketMessageType
    {
        NotifyOpponentMove,
        NotifyGameJoined,
        NotifyGameCreated,
        NotifyOpponentJoin,
        NotifyOpponentLeft,
        NotifyUsernameChanged
    }

    [ServiceBehavior(InstanceContextMode = InstanceContextMode.PerSession, ConcurrencyMode =
ConcurrencyMode.Multiple)]
    public class Game : IGame
    {
        private static Dictionary<string, IGameCallback> _netHttpClientsLobby = new
                Dictionary<string, IGameCallback>();
        private static Dictionary<string, IGameCallback> _webSocketClientsLobby = new
                Dictionary<string, IGameCallback>();

        private static List<HostedGame> _gamesList = new List<HostedGame>();

        public void Subscribe(string clientName, bool isWebSocketClient)
        {
            // Subscribe the client to the game lobby
            var callback = OperationContext.Current.GetCallbackChannel<IGameCallback>();
            bool usernameChanged = false;

            if (callback != null)
            {
                if (_netHttpClientsLobby.ContainsKey(clientName) ||
                            _webSocketClientsLobby.ContainsKey(clientName))
                {
                    clientName = clientName + DateTime.UtcNow.Millisecond;
                    usernameChanged = true;
                }
                lock (_netHttpClientsLobby)
                {
                    if (!isWebSocketClient)
                        _netHttpClientsLobby.Add(clientName, callback);
                    else
                        _webSocketClientsLobby.Add(clientName, callback);
                }
            }
            // NOTIFY USER WHEN USERNAME IS CHANGED
            if (usernameChanged)
            {
                // NETHTTP CLIENTS
                foreach (var gameCallback in _netHttpClientsLobby.Where(
                            gameCallback => clientName == gameCallback.Key))
                {
                    gameCallback.Value.NotifyUsernameChanged(clientName);
                }

                // WEBSOCKET CLIENTS
                foreach (var gameCallback in _webSocketClientsLobby.Where(
                            gameCallback => clientName == gameCallback.Key))
                {
                    gameCallback.Value.SendMessage(CreateMessage(
                            SocketMessageType.NotifyUsernameChanged.ToString()
                            + ";" + clientName));
```

```
            }
        }
    }

    public void Unsubscribe(string clientName)
    {
        // Remove NetHTTP client
        if (_netHttpClientsLobby.ContainsKey(clientName))
            _netHttpClientsLobby.Remove(clientName);

        // Remove webSocket client
        else if (_webSocketClientsLobby.ContainsKey(clientName))
            _webSocketClientsLobby.Remove(clientName);

        foreach (var game in _gamesList.ToList())
        {
            // Remove game if client is host or opponent
            if (game.Host != clientName && game.Opponent != clientName) continue;
            EndGame(game.GameId);
        }
    }

    public void CreateGame(string clientName, int rows, int columns)
    {
        var game = new HostedGame
                       {
                           // GameId = tictac_33PerttiKeinonen
                           GameId = "tictac_" + rows + columns + clientName,
                           Host = clientName,
                           Opponent = null,
                           Round = 0,
                           WaitingForOpponent = true,
                           Columns = columns,
                           Rows = rows
                       };
        lock (_gamesList)
        {
            _gamesList.Add(game);
        }

        // NETHTTP CLIENTS
        foreach (var gameCallback in _netHttpClientsLobby.Where(
           gameCallback => clientName == gameCallback.Key))
        {
            gameCallback.Value.NotifyGameCreated(game.GameId);
            return;
        }

        // WEBSOCKET CLIENTS
        foreach (var gameCallback in _webSocketClientsLobby.Where(
           gameCallback => clientName == gameCallback.Key))
        {
            gameCallback.Value.SendMessage(CreateMessage(
                        SocketMessageType.NotifyGameCreated.ToString()
                        + ";;;;;" + game.GameId));
            return;
        }
    }

    public void JoinGame(string opponentName, int rows, int columns)
    {
        var callback = OperationContext.Current.GetCallbackChannel<IGameCallback>();
        if (callback == null) return;

        string foundOpponent = null;
        string foundGame = null;

        // FIND GAME THAT IS WAITING FOR OPPONENT
        foreach (var hostedGame in _gamesList.Where(
                        hostedGame => hostedGame.WaitingForOpponent))
        {
            // Find available game by gameboard size
            if (rows == hostedGame.Rows && columns == hostedGame.Columns)
            {
                hostedGame.Opponent = opponentName;
                hostedGame.WaitingForOpponent = false;
```

```
                foundGame = hostedGame.GameId;
                foundOpponent = hostedGame.Opponent;

                // NETHTTP CLIENTS
                foreach (var gameCallback in _netHttpClientsLobby.ToList())
                {
                    if (hostedGame.Host == gameCallback.Key)
                    {
                        // NOTIFY for Host
                        gameCallback.Value.NotifyOpponentJoin(hostedGame.Opponent);
                    }

                    if (opponentName == gameCallback.Key)
                    {
                        // NOTIFY for opponent
                        gameCallback.Value.NotifyGameJoined(
                          hostedGame.GameId, hostedGame.Host);
                    }
                }

                // WEBSOCKET CLIENTS
                foreach (var gameCallback in _webSocketClientsLobby.ToList())
                {
                    if (hostedGame.Host == gameCallback.Key)
                    {
                        // NOTIFY for Host
                        gameCallback.Value.SendMessage(CreateMessage(
                          SocketMessageType.NotifyOpponentJoin.ToString()
                          + ";;;;;" + hostedGame.Opponent));
                    }

                    if (opponentName == gameCallback.Key)
                    {
                        // NOTIFY for opponent
                        gameCallback.Value.SendMessage(CreateMessage(
                          SocketMessageType.NotifyGameJoined.ToString() +
                           ";;;;" + hostedGame.GameId + ";" + hostedGame.Host));
                    }
                }
                return;
            }
        }

        // IF GAME NOT FOUND
        if (foundGame == null || foundOpponent == null)
        {
            var gameCallback = _netHttpClientsLobby.First(
                        client => client.Key == opponentName);
            gameCallback.Value.NotifyGameJoined(null, null);

            var gameSocketCallback = _webSocketClientsLobby.First(
                        client => client.Key == opponentName);
            gameSocketCallback.Value.SendMessage(CreateMessage(
                        SocketMessageType.NotifyGameJoined.ToString() + ";" + ""));
        }
    }

    public void EndGame(string gameId)
    {
        var callback = OperationContext.Current.GetCallbackChannel<IGameCallback>();

        if (callback == null) return;

        foreach (var hostedGame in _gamesList.ToList())
        {
            if (gameId != hostedGame.GameId) continue;

            // NETHTTP CLIENTS
            foreach (var gameCallback in _netHttpClientsLobby.ToList())
            {
                if (hostedGame.Opponent == gameCallback.Key)
                {
                    gameCallback.Value.NotifyOpponentLeft(hostedGame.Host);
                }
                else if (hostedGame.Host == gameCallback.Key)
                {
```

```
                    gameCallback.Value.NotifyOpponentLeft(hostedGame.Opponent);
                }
            }

            // WEBSOCKET CLIENTS
            foreach (var gameCallback in _webSocketClientsLobby.ToList())
            {
                if (hostedGame.Opponent == gameCallback.Key)
                {
                    gameCallback.Value.SendMessage(CreateMessage(
                          SocketMessageType.NotifyOpponentLeft.ToString()
                          + ";" + hostedGame.Host));
                }
                else if (hostedGame.Host == gameCallback.Key)
                {
                    gameCallback.Value.SendMessage(CreateMessage(
                          SocketMessageType.NotifyOpponentLeft.ToString()
                          + ";" + hostedGame.Opponent));
                }
            }
            _gamesList.Remove(hostedGame);
        }
    }

    public void SendPlayerMove(string player, string opponent, int row, int column)
    {
        var callback = OperationContext.Current.GetCallbackChannel<IGameCallback>();

        if (callback == null) return;

        // NETHTTP CLIENTS
        foreach (KeyValuePair<string, IGameCallback> client in _netHttpClientsLobby.Where(
           client => client.Key.Equals(opponent)))
        {
            client.Value.NotifyOpponentMove(player, opponent, row, column);
        }

        // WEBSOCKET CLIENTS
        foreach (KeyValuePair<string, IGameCallback> client in _webSocketClientsLobby.Where(
           client => client.Key.Equals(opponent)))
        {
            client.Value.SendMessage(CreateMessage(
                          SocketMessageType.NotifyOpponentMove.ToString()
                          + ";" + player + ";" + row + ";" + column + ";" + opponent));
        }
    }

    public void UnidentifiedMessage(Message message)
    {
        if (message == null)
        {
            throw new ArgumentNullException("message");
        }

        var property = (WebSocketMessageProperty)message.Properties[
                          "WebSocketMessageProperty"];

        var context = property.WebSocketContext;
        var queryParameters = HttpUtility.ParseQueryString(context.RequestUri.Query);
        string content = string.Empty;

        if (!message.IsEmpty)
        {
            byte[] body = message.GetBody<byte[]>();
            content = Encoding.UTF8.GetString(body);
        }

        string str = null;
        if (string.IsNullOrEmpty(content))
        {
            str = queryParameters["Name"];
            Subscribe(str, true);
        }
        else
        {
            // Action;ClientName;Row;Column;Opponent;GameId
```

```
                // ACTION(Create,Join,Move,End)
                string[] splittedContent = content.Split(';');
                int row, column;

                switch (splittedContent[0])
                {
                    case "create":
                        if (int.TryParse(splittedContent[2], out row) && int.TryParse(splitted-
Content[3], out column))
                            CreateGame(splittedContent[1], row, column);
                        break;

                    case "join":
                        if (int.TryParse(splittedContent[2], out row) && int.TryParse(splitted-
Content[3], out column))
                            JoinGame(splittedContent[1], column, row);
                        break;

                    case "move":
                        if (int.TryParse(splittedContent[2], out row) && int.TryParse(splitted-
Content[3], out column))
                        SendPlayerMove(splittedContent[1], splittedContent[4], row, column);
                        break;

                    case "end":
                        EndGame(splittedContent[1]);
                        break;

                    case "unsub":
                        Unsubscribe(splittedContent[1]);
                        break;
                }
            }
        }

        private Message CreateMessage(string content)
        {
            var message = ByteStreamMessage.CreateMessage(new ArraySegment<byte>(
                            Encoding.UTF8.GetBytes(content)));
            message.Properties["WebSocketMessageProperty"] = new WebSocketMessageProperty {
                            MessageType = WebSocketMessageType.Text };

            return message;
        }
    }
}
```

WCF CHATSERVICE WEB.CONFIG Liite 3

```
<?xml version="1.0"?>
<configuration>
  <!--  To collect diagnostic traces, uncomment the section below or merge with existing system.diagnostics section.
        To persist the traces to storage, update the DiagnosticsConnectionString setting with your storage credentials.
        To avoid performance degradation, remember to disable tracing on production deployments.
  <system.diagnostics>
    <sharedListeners>
      <add name="AzureLocalStorage" type="ChatService.AzureLocalStorageTraceListener, ChatService"/>
    </sharedListeners>
    <sources>
      <source name="System.ServiceModel" switchValue="Verbose, ActivityTracing">
        <listeners>
          <add name="AzureLocalStorage"/>
        </listeners>
      </source>
      <source name="System.ServiceModel.MessageLogging" switchValue="Verbose">
        <listeners>
          <add name="AzureLocalStorage"/>
        </listeners>
      </source>
    </sources>
   </system.diagnostics> -->
  <system.diagnostics>
    <trace>
      <listeners>
        <add type="Microsoft.WindowsAzure.Diagnostics.DiagnosticMonitorTraceListener, Microsoft.WindowsAzure.Diagnostics, Version=1.8.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35"
          name="AzureDiagnostics">
          <filter type="" />
        </add>
      </listeners>
    </trace>
  </system.diagnostics>
  <system.web>
    <compilation debug="true" targetFramework="4.5" />
    <httpRuntime targetFramework="4.5"/>
  </system.web>
  <system.serviceModel>

    <services>
      <service name="ChatService.WCFDuplexService">
        <endpoint address="" binding="netHttpBinding" contract="ChatService.IChatRoom"/>
        <endpoint address="mex" binding="mexHttpBinding" contract="IMetadataExchange"/>
        <endpoint address="phoneSocket" binding="customBinding" bindingConfiguration="webSocket"
        contract="ChatService.IChatRoom" />
      </service>
    </services>

    <protocolMapping>
      <add scheme="http" binding="netHttpBinding" bindingConfiguration="netHttp"/>
    </protocolMapping>

    <bindings>
      <customBinding>
        <binding name="webSocket">
          <byteStreamMessageEncoding/>
          <httpTransport>
            <webSocketSettings transportUsage="Always"
               createNotificationOnConnection="true"/>
          </httpTransport>
        </binding>
      </customBinding>

        <netHttpBinding>
        <binding name="netHttp" openTimeout="00:10:00"
              receiveTimeout="00:10:00" closeTimeout="00:10:00">
          <webSocketSettings transportUsage="WhenDuplex"/>
          <security mode="None" />
```

```
        </binding>
      </netHttpBinding>
    </bindings>

    <behaviors>
      <serviceBehaviors>
        <behavior>
          <!-- To avoid disclosing metadata information, set the value below to false
               before deployment -->
          <serviceMetadata httpGetEnabled="true"/>
          <!-- To receive exception details in faults for debugging purposes, set the
               value below to true.  Set to false before deployment to avoid disclosing
               exception information -->
          <serviceDebug includeExceptionDetailInFaults="true"/>
        </behavior>
      </serviceBehaviors>
    </behaviors>

    <serviceHostingEnvironment multipleSiteBindingsEnabled="true" />
  </system.serviceModel>
  <system.webServer>
    <modules runAllManagedModulesForAllRequests="true"/>
    <!--
        To browse web app root directory during debugging, set the value below to true.
        Set to false before deployment to avoid disclosing web app folder information.
    -->
    <directoryBrowse enabled="true"/>
  </system.webServer>

</configuration>
```