

.NET Enterprise Applications

Roland Hediger

November 6, 2013

Contents

| | | |
|----------|---|-----------|
| 1 | Basics,Intro,TFS | 5 |
| 1.1 | TFS Setup | 5 |
| 2 | Windows Presentation Foundation (WPF) | 6 |
| 2.1 | Goals of WPF | 6 |
| 2.2 | WPF Features | 6 |
| 2.2.1 | Rendering | 6 |
| 2.3 | XAML | 6 |
| 2.3.1 | User Controls vs Custom Control | 9 |
| 2.3.2 | Layout Panels , Padding and Alignment | 9 |
| 2.3.3 | Transformations (Translations/Animations) | 10 |
| 2.4 | Databinding with XAML | 10 |
| 2.4.1 | Data Context | 10 |
| 2.4.2 | INotifyPropertyChanged | 11 |
| 2.4.3 | ObservableCollection | 11 |
| 2.4.4 | Value Converters | 11 |
| 2.4.5 | Problems | 12 |
| 2.5 | MVVM Pattern | 12 |
| 2.5.1 | Commands | 12 |
| 3 | Entity Framework | 14 |
| 3.1 | ADO.NET | 14 |
| 3.1.1 | Examples | 15 |
| 3.2 | Intro to Entity Framework | 15 |
| 3.3 | Architecture / Structure of EF | 15 |
| 3.4 | "Variations" of Entity Framework | 16 |
| 3.5 | Entity Objects | 16 |
| 3.6 | First Steps | 17 |
| 4 | Dependency Injection with Unity | 19 |
| 4.1 | What is a dependency? | 19 |
| 4.1.1 | Tightly Coupled Dependencies (Bad) | 19 |
| 4.2 | What is Inversion of Control | 20 |
| 4.2.1 | IOC Implications | 20 |
| 4.2.2 | IOC Advantages | 20 |
| 4.3 | What is dependency injection | 20 |
| 4.3.1 | Dependency Options | 20 |
| 4.3.2 | Ways to inject dependencies | 20 |
| 4.4 | Pros and Cons of DI | 21 |
| 4.5 | IOC Container Services | 21 |
| 4.6 | IOC based Injection | 21 |
| 4.7 | How to use Unity | 22 |
| 4.7.1 | Resolving Dependencies (Wiring) | 22 |
| 4.8 | Managing Lifetimes of Objects with Unity | 22 |
| 5 | ASP MVC4 | 24 |
| 5.1 | Advantages of MVC | 24 |
| 5.2 | Naming Conventions | 24 |
| 5.3 | MVC Sequence Diagram - How it works | 24 |

| | | |
|----------|--|-----------|
| 5.4 | Controllers | 24 |
| 5.4.1 | Controller Action Types | 25 |
| 5.4.2 | Controllers and AJAX | 25 |
| 5.5 | Passing Data between Controller and View | 25 |
| 5.5.1 | Viewbag | 26 |
| 5.6 | Razor View Engine | 26 |
| 5.7 | View Helpers | 27 |
| 5.8 | Layout | 27 |
| 5.9 | Routing | 27 |
| 5.10 | Model binding | 28 |
| 5.10.1 | Hidden values | 29 |
| 5.10.2 | Step by Step behind the curtain | 29 |
| 5.11 | Sessions | 29 |
| 6 | Azure | 30 |
| 6.1 | Data Center Architecture | 30 |
| 6.2 | Windows Azure Portal | 31 |
| 6.2.1 | PAAS in depth for Azure | 31 |
| 6.3 | Databases/Storage Account | 31 |
| 6.3.1 | Storage Types (Table,Queue) | 32 |

1 Basics,Intro,TFS

1

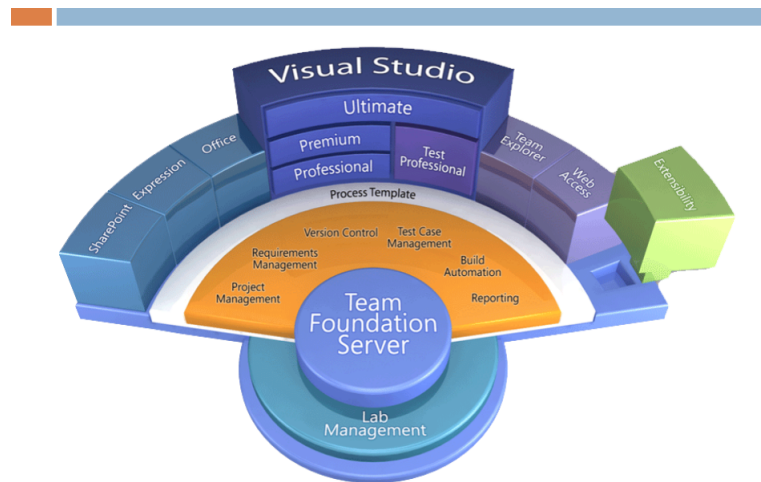


Figure 1.1: TFS Features

1.1 TFS Setup

On Premises • Server Setup

- Server Unterhalt + Backup
- Reporting (SQL Reporting Dienst)
- Sharepoint
- **Voll konfigurierbar im vergleich mit TFS**

TFS als SAAS Lösung • Kein Server Setup

- Kein Unterhalt + Backup
- Saklierbar
- Data Storage off premises.

Additional TFS Features • Sprint Planning

- Scrum Board
- Excel Reporting

General Dont waste too much time choosing the right process template. Out of the box experience is sufficient :
Source Control, Work Item Types, Basic Reporting

¹Author welcomes tokens of gratitude in the form of beer, if thy deem this helpful for thine exam

2 Windows Presentation Foundation (WPF)

2.1 Goals of WPF

1. Unified approach to UI Docs and Media, replacing the individual technologies GDI, GDI+, Win32 (Winforms)
2. **Integrated vector-based composition engine** - One graphics engine for whole stack.
3. Declarative programming : Separate UI look and feel (XAML) from programming (Code Behind).
4. Ease of deployment : Allowing administrators to deploy and manage applications securely.

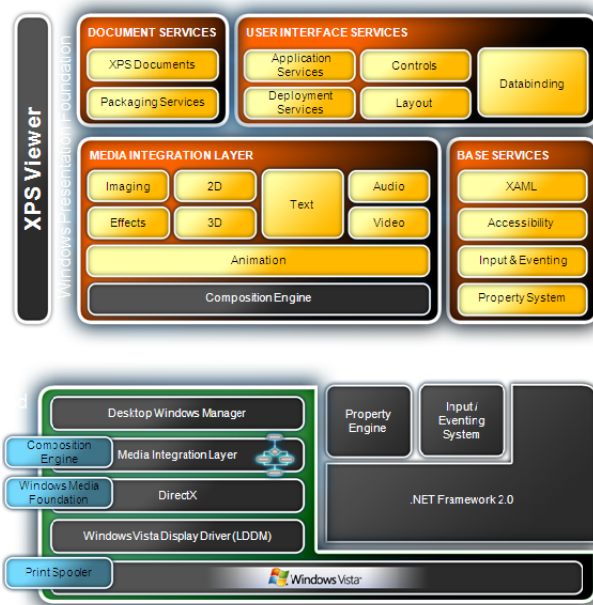


Figure 2.1: WPF Architecture

2.2 WPF Features

2.2.1 Rendering

2.3 XAML

Extensible Application Markup Language, xml based to *instantiate and initialize objects with heirachichal relationships*

Listing 2.1: First XAML Example

```
<Window xmlns="http://schemas.microsoft.com/winfx/..."> <StackPanel
  HorizontalAlignment="Center" > <Image Source="Images/hello.jpg"
  Height="80" /> <TextBlock Text="Welcome to WPF!" FontSize="14"/> <Button
  Content="OK" Padding="10,4" /> </StackPanel> </Window>
```

- ❑ Completely vector-based
- ❑ DirectX
- ❑ Hardware acceleration of GPUs

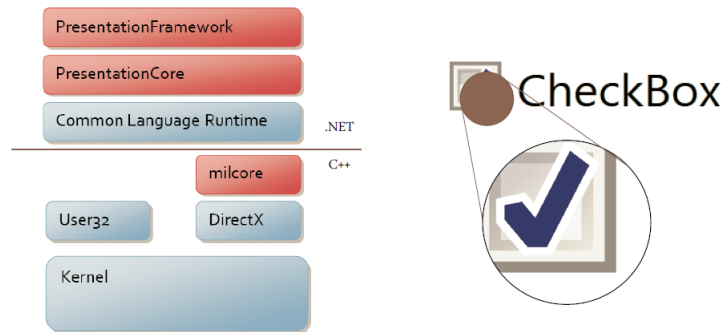


Figure 2.2: figure

General Markup for windows : Build applications in simple declarative statements. Can be used for any CLR object heirarchy.Code and content are strictly separate, streamlines designer developer collaboration.

XAML vs Code Everything in XAML can also be done in code. The xml tags correspond to objects using their default constructors, and xml attributes correspond to the UI Object Properties.

```
<StackPanel>
  <TextBlock Margin="20">Hello</TextBlock>
</StackPanel>
```

XML

```
StackPanel stackPanel = new StackPanel();

TextBlock textBlock = new TextBlock();
textBlock.Margin = new Thickness(10);
textBlock.Text = "Welcome to WPF";
stackPanel.Children.Add(textBlock);
```

C#

Figure 2.3: xaml vs c#

XAML Prefixes + Property Element Syntax

Listing 2.2: XAML Prefixes

```
\begin{lstlisting} [caption=XAML Property Element Syntax]
<Rectangle Width="20" Height="20"> <Rectangle.Fill>
  <LinearGradientBrush> <GradientStop Color="Red" Offset="0" />
  <GradientStop Color="Blue" Offset="1" /> </LinearGradientBrush>
</Rectangle.Fill> </Rectangle>
```

XAML Compilation

UI Services

- Layout
- Controls Library
- Templates

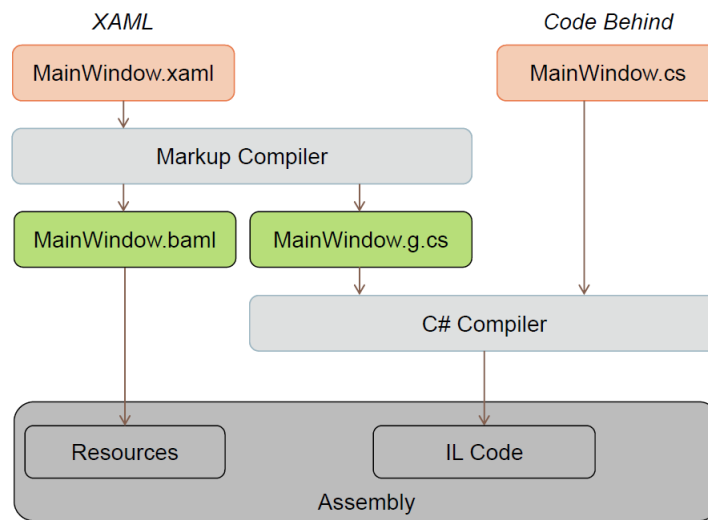


Figure 2.4: figure

- Styles and Resources

Listing 2.3: Combination of UI Services Example

```

<StackPanel>
<StackPanel.Triggers>
<EventTrigger RoutedEvent="Button.Click">
<EventTrigger.Actions>
<BeginStoryboard>
<BeginStoryboard.Storyboard>
<Storyboard>
<ColorAnimation To="Yellow" Duration="0:0:0.5"
Storyboard.TargetName="TheBrush"
Storyboard.TargetProperty="Color" />
<DoubleAnimation To="45" Duration="0:0:2"
Storyboard.TargetName="LowerEllipseTransform"
Storyboard.TargetProperty="Angle" />
...
</StackPanel.Triggers>
... remainder of contents of StackPanel, including x:Name'd
TheBrush and LowerEllipseTransform ...
</StackPanel>
  
```

Flexible Composition I can define an Item inside the content tag of an Item and it will be used as the parent item's content :

Listing 2.4: XAML Composition Example

```

<Button Width="50">
<Button.Content>
<Image Source="images/windows.jpg"
Height="40"/>
</Button.Content>
</Button>
  
```

Attached Properties Allows a child element of an object to adjust properties of itself in relation to the parent object (where it should dock, margins) which are only available due to the type of the parent element.

Listing 2.5: XAML Attached Properties Example

```

<DockPanel>
<Button DockPanel.Dock="Left" Content="Button" />
</DockPanel>
<Canvas>
<Button Canvas.Top="20" Canvas.Left="20"
Content="Button" />
</Canvas>

```

2.3.1 User Controls vs Custom Control

User controls are reusable compositions of other controls : Grid with items positioned on it the same way used many times over.

CustomControl Self explanatory - enhances existing control.

2.3.2 Layout Panels , Padding and Alignment

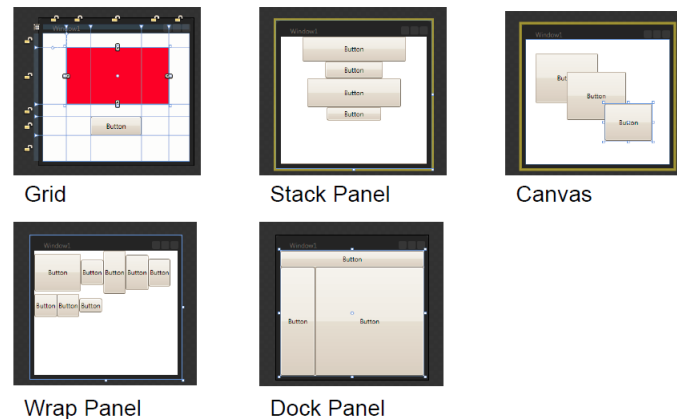
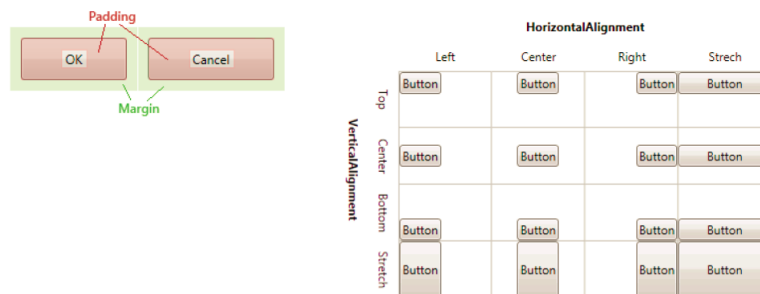


Figure 2.5: WPF Panels



```

<Button HorizontalAlignment="Left"
        VerticalAlignment="Top"
        Margin="8,0,8,8" >
    Test
</Button>

```

Figure 2.6: Padding and Alignment

- Use Alignment, Padding and Margin to position elements
- Avoid fix sizes for elements
- Do not misuse the Canvas Panel for fix positioning of elements (WinForms Style)

2.3.3 Transformations (Translations/Animations)

Element can be transformed in WPF LayoutTransform influences the layout, RenderTransform does not.

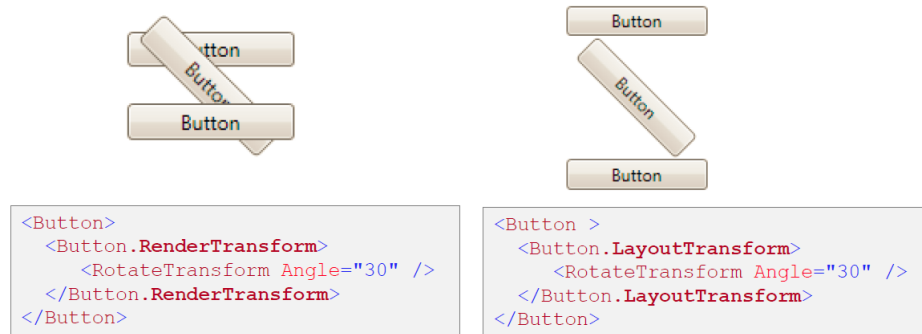


Figure 2.7: figure

2.4 Databinding with XAML

```
<TextBlock Text="{Binding Path=Vorname}" />
```

- DataBinding synchronizes the values of two properties.
- Typically UI element is connected to an entity object (POCO / DAO) from a database.
- Binding can be Uni or Bidirectional.
- A ValueConverter can adapt different data formats for synchronization.

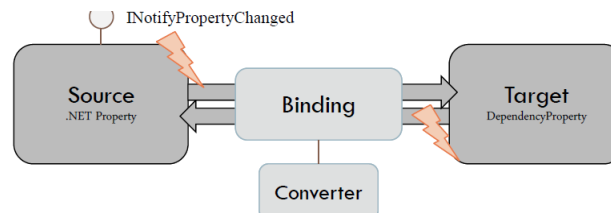


Figure 2.8: Data Binding Diagram

2.4.1 Data Context

Every WPF element has a DataContext property. The DataContext is inherited to children. Allows the Binding to a Data-Object. The default source of a Bindings is always the DataContext.

Listing 2.6: Data Binding Example

```
<Button Content="{Binding Name}" />
DataContext = customer1;
```

Properties:

UpdateSourceTrigger PropertyChanged, LostFocus, Explicit(manual)

Mode Direction of Databinding : OneWay, TwoWay, OneWayToSource.

2.4.2 IPropertyChanged

Every Data-Object must implement INotifyPropertyChanged to allow the propagation of changes.

Listing 2.7: INotifyPropertyChanged implementation

```
public class Customer : INotifyPropertyChanged
{
    private string _name;
    public string Name
    {
        get { return _name; }
        set
        {
            _name = value;
            NotifyPropertyChanged("Name");
        }
    }
    public event PropertyChangedEventHandler PropertyChanged;
    private void NotifyPropertyChanged( string name)
    {
        if( PropertyChanged != null )
            PropertyChanged(this, new PropertyChangedEventArgs(name));
    }
}
```

2.4.3 ObservableCollection

Use ObservableCollection to allow the propagation of collection changes.

```
ObservableCollection<Auction> auctions
= new ObservableCollection<Auction>();
```

2.4.4 Value Converters

ValueConverters can change the format of the data in both directions:

Listing 2.8: ValueConverter Example

```
public class BoolToVisibilityConverter : IValueConverter
{
    #region IValueConverter Members
    public object Convert(object value, Type targetType, object parameter,
        CultureInfo culture)
    {
        return (bool) value ? Visibility.Visible : Visibility.Collapsed;
    }
    public object ConvertBack(object value, Type targetType, object parameter,
        CultureInfo culture)
    {
        throw new NotImplementedException();
    }
    #endregion
}
```

Listing 2.9: ValueConverter in XAML

```
<Window.Resources>
```

```
<conv:BooleanToStatusTextConverter
x:Key="booleanToStatusTextConverter" />
</Window.Resources>
<Button Content="{Binding IsOpen,
Converter={booleanToStatusTextConverter}}" />
```

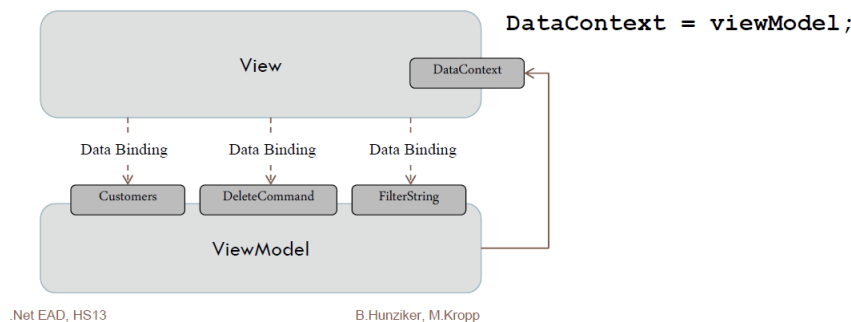
2.4.5 Problems

DataBinding errors are written to the Debug Output. Use an empty ValueConverter to set breakpoints.

2.5 MVVM Pattern

- Motivation**
- Separation of GUI design ("style") and logic "behavior". Ability to use Expression Blend.
 - No duplicated code to update views..No "myLabel.Text = newValue" sprinkled in code behind everywhere.
 - Testability: Since your logic is completely agnostic of your view (no "myLabel.Text" references), unit testing is made easy.

- ▣ Each View binds to a single ViewModel that provides all functionality and data
 - Makes the ViewModel unit-testable
 - Simplifies the data binding



.Net EAD, HS13

B.Hunziker, M.Kropp

6

Figure 2.9: MVVM Example

View(xaml) UserControl based, with xaml. Has minimal code behind. DataContext is set to the associated View Model. No event Handlers. Data binding of view is set to view model.

View Model(c#) Implements INotifyPropertyChanged. Exposes ICommand, handles validation. Functions as an adapter class between view and model, as a result, it is testable.

Model POCO free of all WPF.

2.5.1 Commands

Commands are used to bind UI actions to the ViewModel functionality. A command implements 3 functions of the ICommand interface :

```
Execute(object param);
canExecute(object param);
even CanExecuteChanged;
```

Commands can be used on different UI Elements.

Listing 2.10: ICommand Implementation Example

```
public class FooCommand : ICommand
{
    public Action<object> ExecuteDelegate { get; set; }
    public Func<object, bool> CanExecuteDelegate { get; set; }

    #region ICommand Members

    public bool CanExecute(object parameter)
    {
        return CanExecuteDelegate(parameter);
    }

    public event EventHandler CanExecuteChanged;

    public void Execute(object parameter)
    {
        ExecuteDelegate(parameter);
    }

    #endregion
}
```

3 Entity Framework

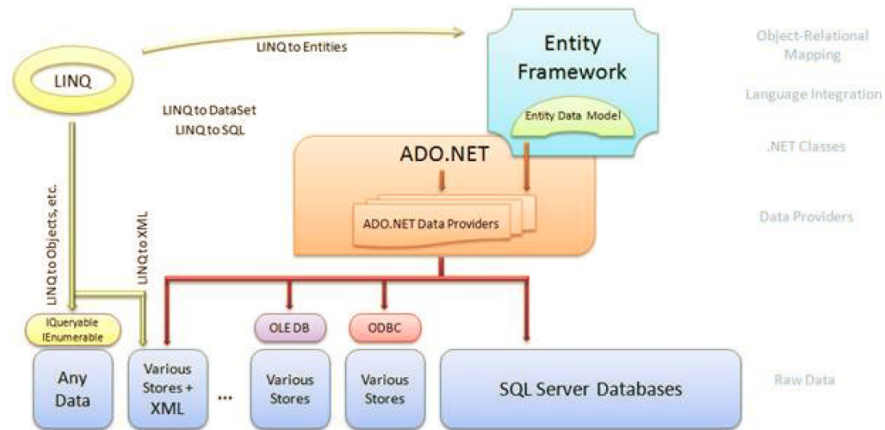


Figure 3.1: Underlying Architecture 1

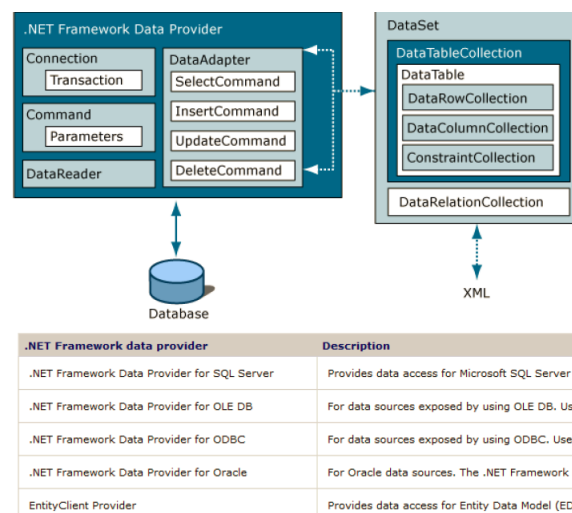


Figure 3.2: Underlying Architecture 2

3.1 ADO.NET

klassenaufistung:

- SqlConnection
- SqlCommand
- SqlCommandBuilder
- SqlDataAdapter
- SqlDataReader

- SqlException
- SqlParameter
- SqlBulkCopy
- SqlTransaction

3.1.1 Examples

Listing 3.1: DataSet Example

```
using (SqlConnection cn = new SqlConnection("<connection string>") {
    cn.Open();
    SqlDataAdapter ad = new OleDbDataAdapter("select * from products",
        cn);
    DataSet ds = new DataSet();
    ad.Fill(ds, "Products");
    foreach(DataRow dr in ds.Tables["Products"].Rows) {
        Console.WriteLine(dr["ProductName"]);
    }
}
```

3.2 Intro to Entity Framework

EF Maps conceptual model to physical tables, uses LINQ. Has features like change tracking identity resolution, lazy loading and more.

LINQ to SQL Lightweight no mapping. Replaced by EF but still supported.

NHibernate Open Source like Entity Framework, uses LINQ.

3.3 Architecture / Structure of EF

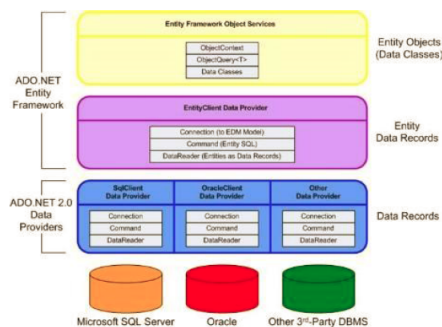


Figure 3.3: EF Architecture

EDMX Files

Object Context (DB Context)

Methods:

`SaveChanges()`

`CreateObjectSet()` (-> LINQ-Query)

`Attach()` / `Detach()`

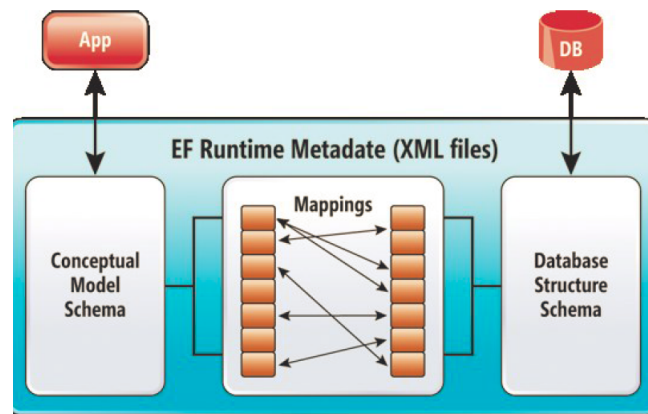


Figure 3.4: EDMX Files

```
<?xml version="1.0" encoding="utf-8"?>
<edmx:Edmx Version="2.0" xmlns:edmx="http://schemas.microsoft.com/ado/2008/10/edmx">
  <!-- EF Runtime content -->
  <edmx:Runtime>
    <!-- SSDL content -->
    <edmx:StorageModels>...</edmx:StorageModels>
    <!-- CSDL content -->
    <edmx:ConceptualModels>...</edmx:ConceptualModels>
    <!-- C-S mapping content -->
    <edmx:Mappings>...</edmx:Mappings>
  </edmx:Runtime>
  <!-- EF Designer content (DO NOT EDIT MANUALLY BELOW HERE) -->
  <Designer xmlns="http://schemas.microsoft.com/ado/2008/10/edmx/Designer">...</Designer>
</edmx:Edmx>
```

Figure 3.5: EDMX Files

```
efresh()
Properties:
Connection
ObjectStateManager
```

Lazy Loading Default, Eager loading - load everything - must be specified explicitly.

StoredProcedure mapping Usually sucks, maps to a single entity with one method to fetch the result of stored procedure, as dataset.

3.4 "Variations" of Entity Framework

Database First : All code for Entities generated from database. (Model)

Code First No .edmx files, DB Schema and Entity Objects / Model from code.

Model first Uses UI Designer, and edmx file with T4 Templating to generate Entity Objects and Context.

3.5 Entity Objects

Notes Important to consider dev performance vs actual runtime performance because ORM means overhead.

Design Considerations • Encapsulate DAL implementation? (technology, connections, queries, structure)

- Technology (ADO.NET vs. EF vs.)
- Logical – physical model mapping?
- Performance & scalability
- Batching: reduce round-trips (performance), Bulk insert/update: for large volumes

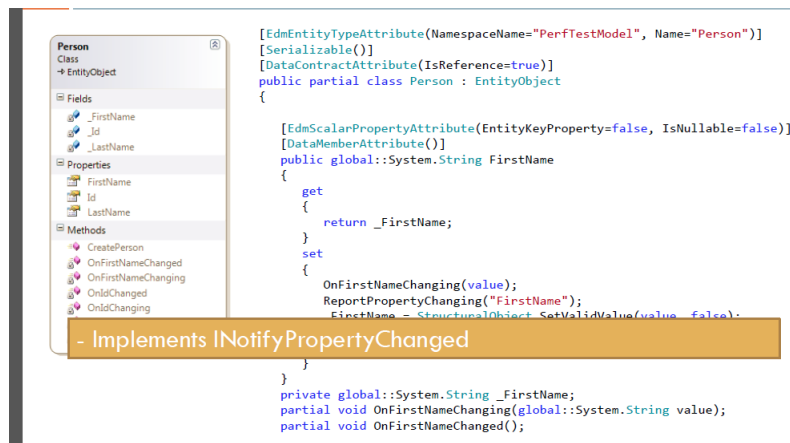


Figure 3.6: Entity Framework Object using Model based approach

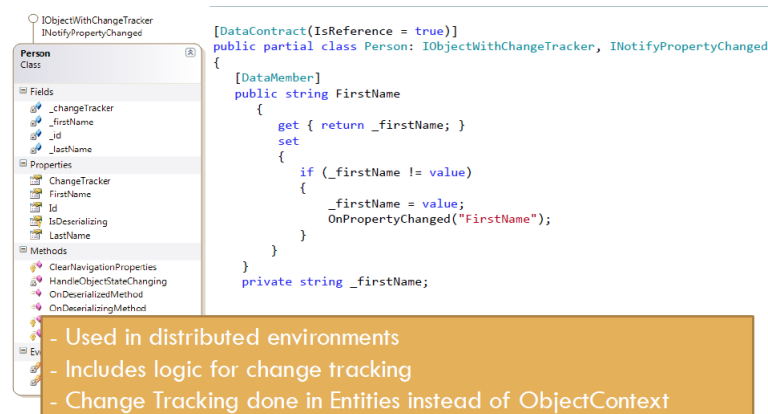


Figure 3.7: Object with Change Tracking

- Use Views?
- Use Stored Procedures?

Benefits of ORM • Decouples logical model from database model

- Consistent, independent query language
- Rich designer support
- Most common patterns (1-to-many, many-to-1, many-to-many, self-referencing, inheritance,
- Slower than ADO.NET core
- Must regenerate EDM after DB changes

3.6 First Steps

Listing 3.2: Initial Entity Framework Example

```
// create dbcontext
AuctionContext context = new AuctionContext();
// get all auctions from database
var myAuctions = context.Auctions;
// get all open auctions
var openAuctions = context.Auctions.Where(a => a.EndTime < DateTime.Now);
// add new auction to database
```



```
var newAuction = new Auction() {  
    context.AddToAuctions(newAuction); context.SaveChanges();  
}
```

4 Dependency Injection with Unity

4.1 What is a dependency?

Dependencies in UML :

Common Dependencies in Enterprise Apps:

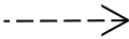





- One
- UML Dependency 
- However there are many kinds
 - Inheritance 
 - Composition 
 - Association 
 - Usage 
 - Create 

Figure 4.1: Dependency examples in UML

Application Layers Data Access Layer & Database dependencies, Dependencies between business layer, and DB Layer.

External Services and Components Web Services, third party components.

.NET Components File Objects / Http Objects (MVC) - HttpContext, Session.

4.1.1 Tightly Coupled Dependencies (Bad)

Listing 4.1: Bad DI Example

```
public class CustomerService : ICustomerService {
    #region ICustomerService Members
    public CustomerDTO GetACustomerFrom(int id)
{
    CustomerDTOMapper dtoMapper = new CustomerDTOMapper();
    CustomerRepository custRepository = new CustomerRepository();
    return dtoMapper.mapFrom(custRepository.getFrom(id));
}
```

Problems :

- Code is tightly coupled,difficult to isolate when testing, difficult to maintain. Changing out components not possible.

Solutions:

- Inversion of Control - Hollywood principle.
- Dependency injection.

4.2 What is Inversion of Control

Higher level modules should not depend on lower level modules. Both should depend on abstractions (interfaces or abstract classes). *Abstractions should not depend on details*

4.2.1 IOC Implications

- Layers, modularization
- Interface based programming.
- Interface in separate package to implementation.
- Implementations fetched through dependency injection.

4.2.2 IOC Advantages

- Increase loose coupling
 - Abstract interfaces don't change
 - Concrete classes implement interfaces
 - Concrete classes easy to throw away and replace
- Increase mobility
- Increase isolation
 - decrease rigidity
 - Increase testability
 - Increase maintainability

4.3 What is dependency injection

Purpose How do we wire up concrete interfaces? DI gives us the ability to inject external dependency into component

DI is a form of IoC, where implementations are passed into an object through constructors/setters/service look-ups, which the object will 'depend' on in order to behave correctly

4.3.1 Dependency Options

Option 1 – Factory User depends on factory, factory depends on destination.

Option 2 – Locator/Registry/Directory The component still controls the wiring. Instantiation Sequence. Dependency on the Locator.

Option 3 – Dependency Injection An assembler controls the wiring

4.3.2 Ways to inject dependencies

Constructor Injection

```
public class CustomerService : ICustomerService
{
    #region DI
    private ICustomerRepository repository;
    private ICustomerDTOMapper mapper;

    public CustomerService(
        ICustomerRepository repository,
        ICustomerDTOMapper mapper)
    {
        this.repository = repository;
        this.mapper = mapper;
    }
}
```

Figure 4.2: figure

```

public class CustomerService : ICustomerService
{
    private ICustomerRepository customerRepository;
    public ICustomerRepository CustomerRepository
    {
        get
        {
            return customerRepository;
        }
        set
        {
            customerRepository = value;
        }
    }
}

```

Figure 4.3: figure

Setter Injection**Method Injection**

```

public class CustomerService : ICustomerService
{
    public ICustomer GetCustomerDetails
    (
        ICustomerRepository repository,
        int id
    )
    {
        ICustomer customer = repository.GetFrom(id);
        return customer;
    }
}

```

Figure 4.4: figure

4.4 Pros and Cons of DI

- + Loosely Coupled
- + Better Testing
- + Allows IOC Container

4.5 IOC Container Services

- Service Locator - Finds implementation of Interface based on wiring.
- Managing lifetime of dependencies. (Singleton, per http request etc)
- Automatic injection.
- Wiring config.

4.6 IOC based Injection

1. TV depends on an object of some type
2. Ask container to resolve that type
3. IoC container creates object
4. Object gets injected into TV
5. TV uses object

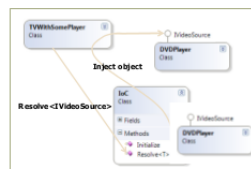


Figure 4.5: IOC based Injection

4.7 How to use Unity

Unity container creates object instances during resolution of types. Container must be asked to resolve a type.

Listing 4.2: DI Unity First Example

```
IUnityContainer container = new UnityContainer();
IVideoSource source = container.resolve<IVideoSource>();
IEnumerable<VideoSource> sources = container.ResolveAll<IVideoSource>();
```

4.7.1 Resolving Dependencies (Wiring)

Dependencies are wired up for resolved types

Properties marked as [Dependency]

Constructor marked [InjectionConstructor]

Methods marked [InjectionMethod]

Unity can also wire up dependencies on objects not created by container

| Wiring Methods : | | Constructor | Properties | Method |
|------------------|--|-------------|------------|--------|
| | | Resolve | Yes | Yes |
| | | BuildUp | No | Yes |

- Container has registration for object resolution Maps interfaces and abstract base classes to
- Concrete types
- Existing object instances
- Performed only once per container instance Can Use code or configuration Register multiple types based on names

```
IUnityContainer container = new UnityContainer();
container
    .RegisterType<TVWithSomePlayer, TVWithSomePlayer>()
    .RegisterType<IVideoSource, DVDPlayer>()
    .RegisterInstance<ILog>(someLogObject);
```

Figure 4.6: Sample Registration using Unity

```
<configSections>
  <section name="unity"
    type="Microsoft.Practices.Unity.Configuration.
      UnityConfigurationSection,
      Microsoft.Practices.Unity.Configuration"/>
</configSections>
```

```
<unity>
  <namespace name="Ploeh.Samples.MenuModel" />
  <assembly name="Ploeh.Samples.MenuModel" />
  <containers>
    <register type="IIngredient" mapTo="Steak" />
  </containers>
</unity>
```

Figure 4.7: Sample Registration in App.config

4.8 Managing Lifetimes of Objects with Unity

- Control the lifetime of objects created by Unity Use LifetimeManager derived classes during registration

- Container controlled: singleton
- Transient: new every time, resolve is called (default)
- Externally managed: keeps weak references
- RegisterInstance method: Singleton not created by container.

5 ASP MVC4

5.1 Advantages of MVC

- A new option for ASP.NET, not a replacement for ASP.NET WebForms
- Simple way to program ASP.NET
- Easy testable
- Much control over your HTML
- Much control over your URLs
- Can be used as a base infrastructure for SPA
- Plays well with others : Entity Framework, Unity, Unit Testing (Single Page Apps)

5.2 Naming Conventions

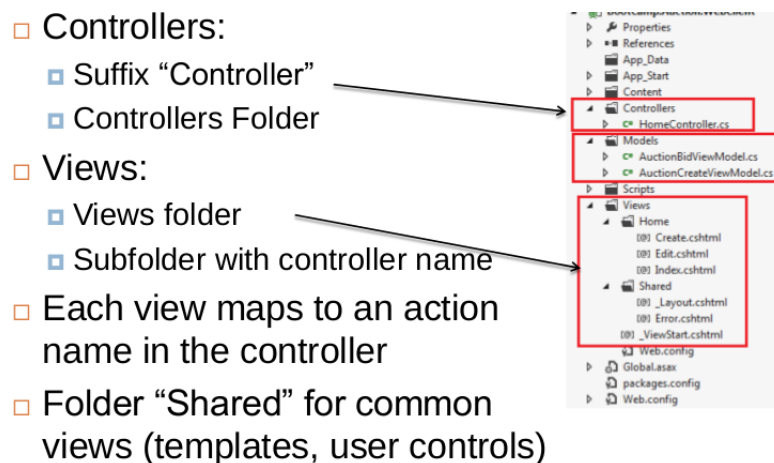


Figure 5.1: MVC Naming Conventions

5.3 MVC Sequence Diagram - How it works

5.4 Controllers

- Controllers handle all incoming requests
- Retrieve data from storage
- Store posted data in storage (http post)
- Pass the data to a View to generate the
- HTML/CSS/JavaScript for display
- Controllers can implement REST APIs (derive controller from ApiController)

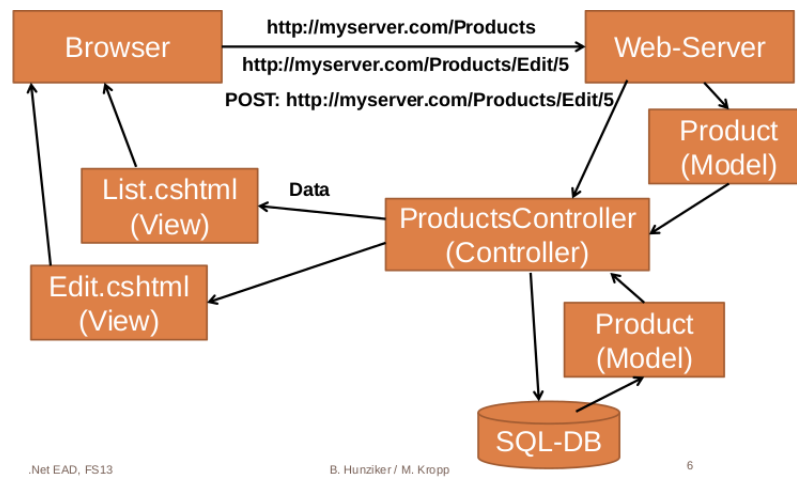


Figure 5.2: MVC Architecture

- The simplest controller just returns HTML to the Web browser
- IMPORTANT: By default, all public methods in a controller class can be called from the Web browser
- Controllers give a `ResultObject` as return value and can be tested without the need for a webserver - Normal object.

5.4.1 Controller Action Types

| Return Type | Content | Example |
|--------------------------------|------------------------|--|
| <code>ViewResult</code> | Html-Page | <code>return View();</code> |
| <code>PartialViewResult</code> | Part of HTML Page | <code>return PartialView();</code> |
| <code>RedirectResult</code> | Redirect to other page | <code>return Redirect("url");</code> |
| <code>FileResult</code> | Binary data | <code>return File("path");</code> |
| <code>JsonResult</code> | Serialized JS Objects | <code>return JSON(MyPOCO);</code> |
| <code>JavaScriptResult</code> | JS (AJAX) | <code>return JavaScript("alert('hi')");</code> |

5.4.2 Controllers and AJAX

Ajax calls can also be handled by the controller These are also public methods:

```

View:
@Ajax.ActionLink("Click me", "Click", null)
Controller:
public JavaScriptResult Click()
{
    return JavaScript("alert('Hallo World');");
}

```

5.5 Passing Data between Controller and View

This part was not taken entirely from the slides because it was badly explained.

1. I can return a view with the `View(Modelobject)` method as follows :

```

public ActionResult Index()
{
    return View(db.Item.ToList());
}

```


This looks for view index.cshtml under views (matches method name). The parameter is the model object. Unless otherwise specified the view expects either no parameter or a parameter of type object, which can be accessed

5.5.1 Viewbag

This is referred to in the slides as the untyped variant. It IS NOT strictly speaking a variant. You are able to send data to the View using a model as above, and fill up the viewbag with objects (properties) at the same time.

Listing 5.1: ViewBag Example

```
ViewBag.Persons = new SelectList(context.People, "Id", "Name");
//In View
@Html.DropDownListFor(model => model.SellerId, (SelectList)ViewBag.Persons)

//Combined Variant
//Controller
public ActionResult Index()
{
Object blab = new Object(); //can be anything
Viewbag.bla = blab
return View(db.Item.ToList());
}

//In View
<h2> Viewbag.blab.toString</h2>
@Html.DropDownListFor(model => model.SellerId, (SelectList)ViewBag.Persons
```

5.6 Razor View Engine

Asp.Net classic view engine can be used but replaced by Razor. Html tags implicitly identified and cause the code block to end. If it doesnt work one can manually force a code block :

Listing 5.2: Normal Razor ASP.NET MVC

```
<h2>Products</h2>
<ul>
    @foreach (var p in products) {
        <li> @p.ProductName </li>
    }
</li>
```

Listing 5.3: Specified Code Block ASP.NET MVC

```
\begin{lstlisting}
//@{code here} html stuff here.
<div>
<b>
@if (HttpContext.Current.Request.IsAuthenticated)
{
@:Change your password:
}
else
{
@:Please change your initial password:
}
</b>
</div>
```

5.7 View Helpers

Methods that you can use in the view to generate HTML tags that correspond with properties of @model.

```
@Html.ActionLink( Edit    Record ,    Edit    , new {Id=3}):
//generates:
<a href=    /Home/Edit/3    >Edit Record</a>
@Html.LabelFor(model => model.FirstName):
//generates:
<label for="FirstName">FirstName</label>

@Html.EditorFor(model => model.FirstName):

//generates
<input class="text-box single-line" data-val="true"
data-val-required="The FirstName field is required."
id="FirstName" name="FirstName" type="text" value="" />
```

5.8 Layout

This was also badly explained in the slides. As we learn in web frameworks we need a template for every view. The template itself is a view in ASP.NET mvc - layout.cshtml. The name for the template is specified in _ViewStart.cshtml, doesn't have to be named layout.

The layout also has access to the Viewbag - Whatever I put in the viewbag in a Controller method, can be displayed in the layout in non dynamic parts, if the property doesn't exist - an empty String is displayed (null) Example :

```
_Layout.cshtml
<html>
<title>@ViewBag.Title</title>
<body>
    @RenderBody()
</body>
</html>

Index.cshtml
@{
    ViewBag.Title = "Your Page Title";
}
<div>Hello World!</div>
```

Figure 5.3: Layout Example

Requirement : Layout must contain @RenderBody that is all

5.9 Routing

Listing 5.4: Routing Example

```
App_Start/RouteConfig.cs
routes.MapRoute(
    "Default", // Route name
    "{controller}/{action}/{id}", // URL with parameters
    new { controller = "Home", action = "Index", id = UrlParameter.Optional }
    // Default
);
```

First route that matches url pattern "wins".

5.10 Model binding

Just like in Spring MVC or JSF the properties of the Model Object can be bound to form fields provided the fields are given suitable names so that the model binder knows what to bind. This is taken care of with the html helpers above. In other words manually making form fields is generally a bad idea.

- The MVC model binder provides a simple way to map posted values to a .Net Framework type passed as an action parameter (Brute-Force!)

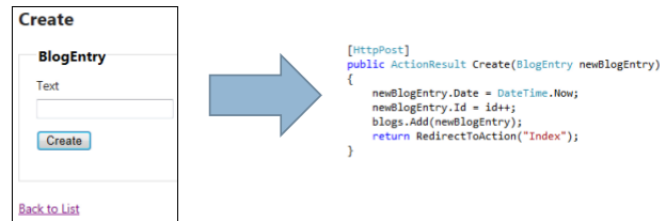


Figure 5.4: Model binder

If you really want to you can extract the values manually from the Request object :

Listing 5.5: Manual reading of form values (not recommended but just in case)

```
public ActionResult Create()
{
    var product = new Product() {
        AvailabilityDate =
            DateTime.Parse(Request["availabilityDate"]),
        CategoryId = Int32.Parse(Request["categoryId"]),
        Description = Request["description"],
        Kind =
            (ProductKind)Enum.Parse(typeof(ProductKind),
            Request["kind"]),
        Name = Request["name"],
        UnitPrice = Decimal.Parse(Request["unitPrice"]),
        UnitsInStock =
            Int32.Parse(Request["unitsInStock"]),
    };
    // do work
}
```

Listing 5.6: Model binding with primitive values

```
public ActionResult Create(
    DateTime availabilityDate, int categoryId,
    string description, ProductKind kind, string name,
    decimal unitPrice, int unitsInStock
)
{
    var product = new Product() {
        AvailabilityDate = availabilityDate,
        CategoryId = categoryId,
```

```

Description = description,
Kind = kind,
Name = name,
UnitPrice = unitPrice,
UnitsInStock = unitsInStock,
};
// do work
}

```

Listing 5.7: Model binding sensibly

```

public ActionResult Create(Product product)
{
    // do work
}

```

5.10.1 Hidden values

@HiddenFor(model=*i*model.property) in the view. Generates a hidden form field that is also mapped to the model later.

5.10.2 Step by Step behind the curtain

Value Sources: querystring parameters, form fields and route data

Evaluation order:

- Previously bound action parameters, when the action is a child action
- Form fields (Request.Form)
- The property values in the JSON Request body (Request.InputStream), but only when the request is an AJAX request
- Route data (RouteData.Values)
- Querystring parameters (Request.QueryString)
- Posted files (Request.Files)

5.11 Sessions

Listing 5.8: using session in controller

```

public ActionResult Index(SessionAspDotNetMvcModel data)
{
    // Assign value into session
    Session["SessionUserID"] = data.sUserID;
    // Redirect view
    return RedirectToAction("EmployeeSection");
}

```

Session States :

InProc: In memory on the Web server (default)

StateServer: Separate process called the ASP.NET state service

SQLServer: SQL Server database Custom: Custom storage provider

Off

6 Azure

Cloud service : On demand , self service, and runs on pay per use principal. Why cloud : Outsourcing of infrastructure, scalability.

Cloud Service Types:

IaaS: Infrastructure as a Service - Virtual Machines (Virtual Private Server)

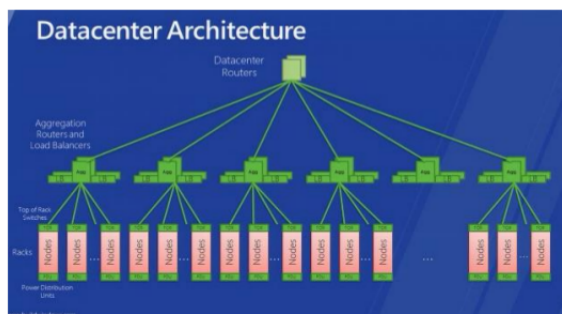
PaaS: Platform as a Service -

- Mobile Services
- Cloud Services
- SQL Databases & Reporting
- Storage
- Service Bus
- Virtual Networks
- Web Sites

Important Characteristics : Usually automates deployment with possibilities for customization through scripts that you can add to the deployment process. Services like openshift for example deploy when you commit your project to the git repo.

[SaaS:] Software as a Service : Site builders and so on fully managed.

6.1 Data Center Architecture



.Net EAD, FS13

- Routers
- Load balancer
- Rack switches, Power switch
- Fabric Controller and Fabric Agent



B. Hunziker / M. Kropp

11

Figure 6.1: figure

6.2 Windows Azure Portal

Service Management Portal UI Subscriptions (roles, URL, billing account) Hosted services, storage, SQL Azure and all the other services. Staging, production, updates. Create, configure, scale, monitor.
Built on top Azure Service Management. REST APIs.
Multiple service administrators.
Get detailed billing information

Modes Free - Limited VM
Shared VM - \$9.68
Standard : Separate VM - \$75 to \$298

PAAS Websites Visual Studio helps, use publish button to send to Azure.

IAAS Features To migrate legacy applications. Full control over the OS Image. Create your VHD locally and upload
Deploy a service package that references and uses the custom OS image
You maintain the OS (patch) With your tools.

6.2.1 PAAS in depth for Azure

- Auto scaling possible Use Azure Diagnostics, avoid logging to local discs
- Azure may “reimage” the VM any time (after patch) -> this removes all local data!
- Production & staging environment
- Virtual IP address swap (easy testing in staging, quick undo possible)

Implementation

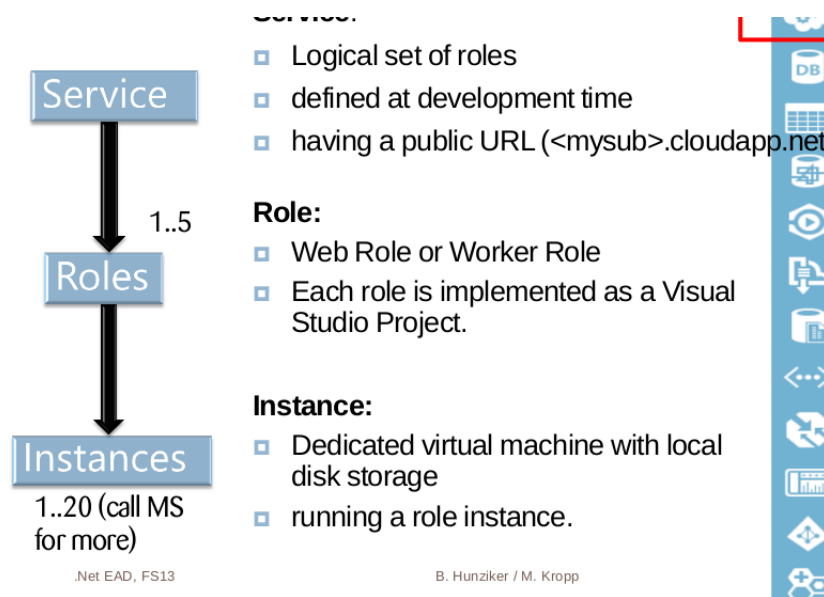


Figure 6.2: figure

6.3 Databases/Storage Account

- Familiar SQL Server Support for existing APIs & tools

- Failover cluster, data stored on 3 different backend data nodes 150 GB limit - “sharding”
- No point in time backup!
- Accessible with Management Studio

Data Sync:

- Synchronize “cloud<>on-premise” data, geo-replicate data
- SQL Server, SQL Azure, SQL Compact (based on .NET Sync Framework)

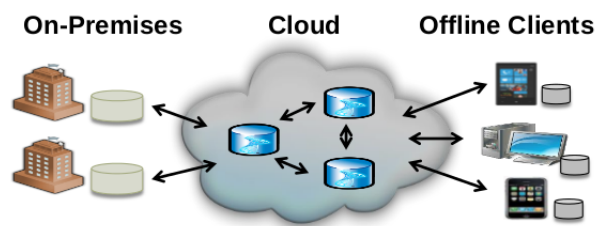


Figure 6.3: Database Syncing

Storage in DB :

- blobs¹, Tables & Queues
- Georeplication (immediately consistent)
- 100TB per storage account
- Accessible via RESTful Web Service API Security: shared secret keys, HTTPS endpoint

6.3.1 Storage Types (Table,Queue)

- Table**
- Rows of entities
 - Up to 255 properties per row
 - Max. 1 MB per entity
 - Partitioned by (partition) key
 - Indexed by row key
 - Scales for large number (billions) of entities
 - Structured data, no fixed schema, not an RDBMS.
- Queue**
- Any binary object can be queued (64kB limit)
 - Use for Inter-role communication
 - non transactional
 - read at least once
 - message invisible for timeout
 - delete to remove message - otherwise is returned to queue

¹Large binary - 200gb