Hosting

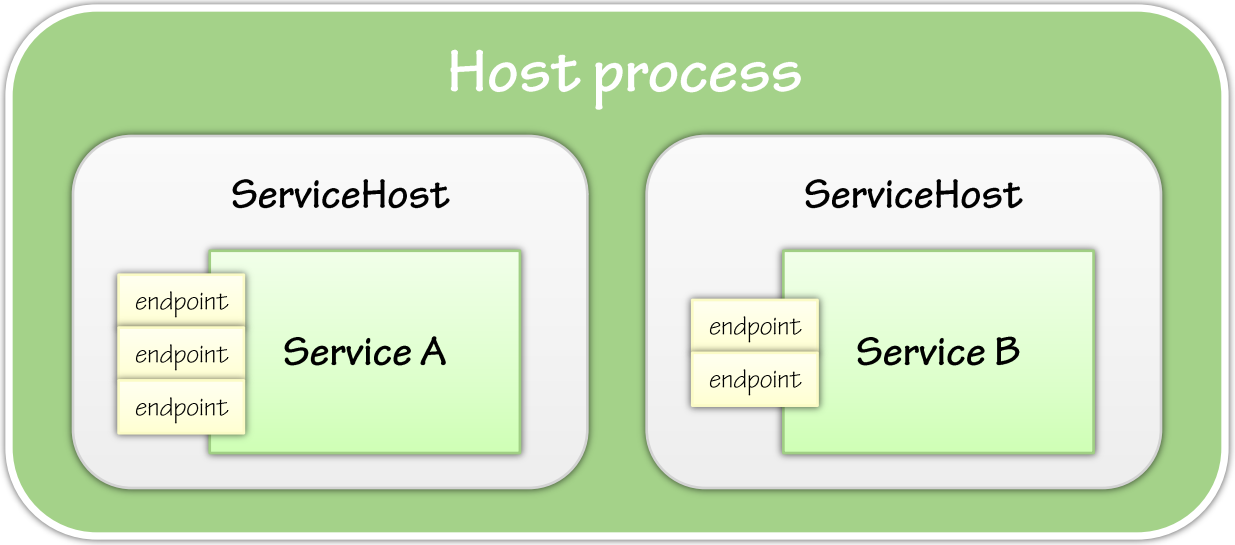
Where should my services run?

Outline

- WCF hosting concepts
- Self-hosting drill-down
- Hosting in Windows Services
- Hosting in IIS 5/6
- Hosting in IIS 7 with WAS

WCF hosting concepts

- **ServiceHost is the WCF programming model for hosting services**
  - A given ServiceHost instance manages a single service type
  - ServiceHost can be used in any host process within a CLR appdomain
  - A host process may contain multiple ServiceHost instances



WCF hosting techniques

- **You can host WCF services in your own applications**
  - Referred to as self-hosting
  - Requires you to manage the lifecycle of the ServiceHost instance
- **Or you can let IIS/ASP.NET manage hosting your services**
  - Referred to as managed hosting
  - ASP.NET manages the ServiceHost instance lifecycle for you

Self-hosting scenarios

- **When does it make sense to use self-hosting techniques?**
  - When you need to host services in clients for callbacks for P2P
  - When you need to take advantage of inproc hosting
  - When you want to avoid IIS for whatever reason
  - When you want to use advanced hosting extensibility points

ServiceHost lifecycle

- **You construct the ServiceHost and specify the service type**
  - ServiceHost reads configuration from application config file
  - You can also configure the instance programmatically
- **You call Open to create the WCF runtime for the service type**
  - Creates the WCF dispatch runtime & messaging runtime
  - Launches worker threads to monitor incoming messages
  - Allows host application to make blocking calls
- **You call Close to gracefully shutdown the WCF runtime**
  - Waits for calls in progress to complete and closes network stack
  - After Close, the service can no longer receive messages
- **You call Abort to "tear down" runtime and close immediately**

ServiceHost lifecycle example

create ServiceHost

open host
(builds runtime)

blocking call

gracefully close
ServiceHost

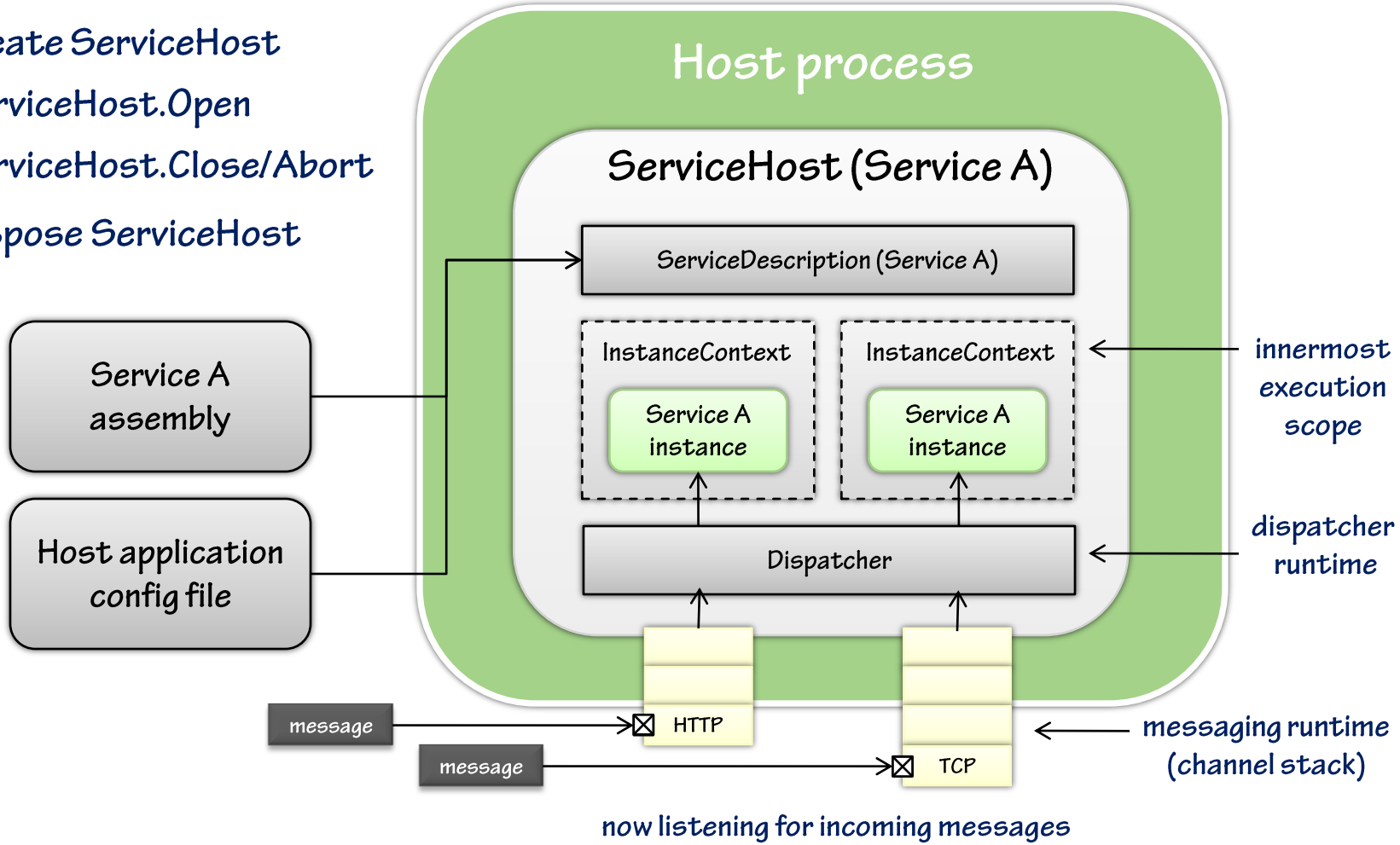
tear-down
ServiceHost

```
class Program {
    static void Main(string[] args) {
        ServiceHost host =
            new ServiceHost(typeof(InvoiceService));
        ... // configure the host before opening

        try {
            host.Open();
            Console.ReadLine();
            host.Close();
        }
        catch (Exception e) {
            Console.WriteLine(e);
            host.Abort();
        }
    }
}
```

ServiceHost architecture

1. Create ServiceHost
2. ServiceHost.Open
3. ServiceHost.Close/Abort
4. Dispose ServiceHost



Host base addresses

- **Each ServiceHost instance can be configured with base addresses**
  - One base address per transport protocol
  - Can be supplied via constructor or configuration section
- **Used to resolve relative endpoint addresses**
  - The binding type tells WCF which base address to use

```
...
ServiceHost host = new ServiceHost(
    typeof(InvoiceService),
    new Uri("http://server:8080/"),
    new Uri("net.tcp://server:8081/"));
host.AddServiceEndpoint(typeof(IInvoiceService),
    new BasicHttpBinding(), "invoiceservice");
...
```

base addresses

relative address

resolves to "http://server:8080/invoiceservice"

Configuring base addresses

configure base addresses per service

relative address

```
<configuration>
  <system.serviceModel>
    <services>
      <service name="InvoiceServiceLibrary.InvoiceService">
        <host>
          <baseAddresses>
            <add baseAddress="http://server:8080/" />
            <add baseAddress="net.tcp://server:8081/" />
          </baseAddresses>
        </host>
        <endpoint address="invoiceservice"
          binding="netTcpBinding"
          contract="InvoiceServiceLibrary.IInvoiceService"/>
      </service>
...
```

resolves to "net.tcp://server:8081/invoiceservice"

Implementing a custom ServiceHost

- **You can implement a custom ServiceHost-derived class**
  - Allows you to customize service configuration (endpoints/behaviors)
  - Allows you to override Dispose
  - Hook lifecycle events: OnOpening, OnOpened, OnClosing, etc.
- **For advanced customization, derive from ServiceHostBase**
  - Requires you to create the internal ServiceDescription

Implementing a custom ServiceHost

override OnOpening

check for behavior

add if it doesn't exist

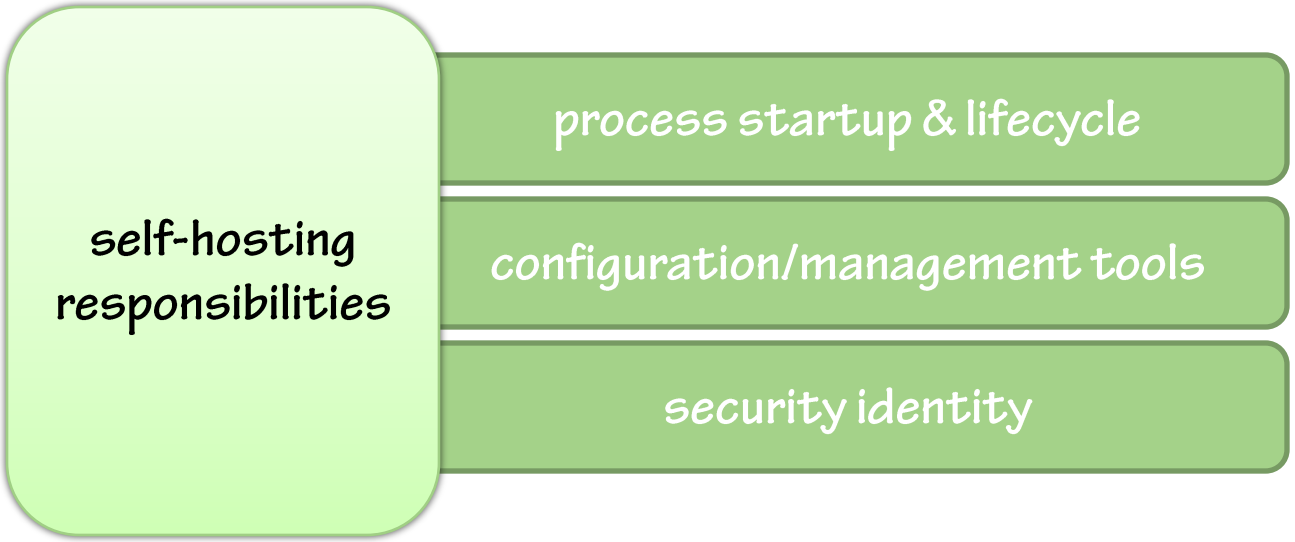
override Dispose

```
public class CustomServiceHost : ServiceHost, IDisposable
{
    public CustomServiceHost(Type serviceType, params Uri[] baseAddresses) :
        base(serviceType, baseAddresses) { }
    public CustomServiceHost(object serviceInstance,
        params Uri[] baseAddresses) : base(serviceInstance, baseAddresses) { }

    protected override void OnOpening()
    {
        base.OnOpening();
        ServiceMetadataBehavior meta =
            this.Description.Behaviors.Find<ServiceMetadataBehavior>();
        if (null == meta)
        {
            meta = new ServiceMetadataBehavior();
            meta.HttpGetEnabled = true;
            this.Description.Behaviors.Add(meta);
        }
    }
    void IDisposable.Dispose()
    {
        ...
```

Self-hosting responsibilities

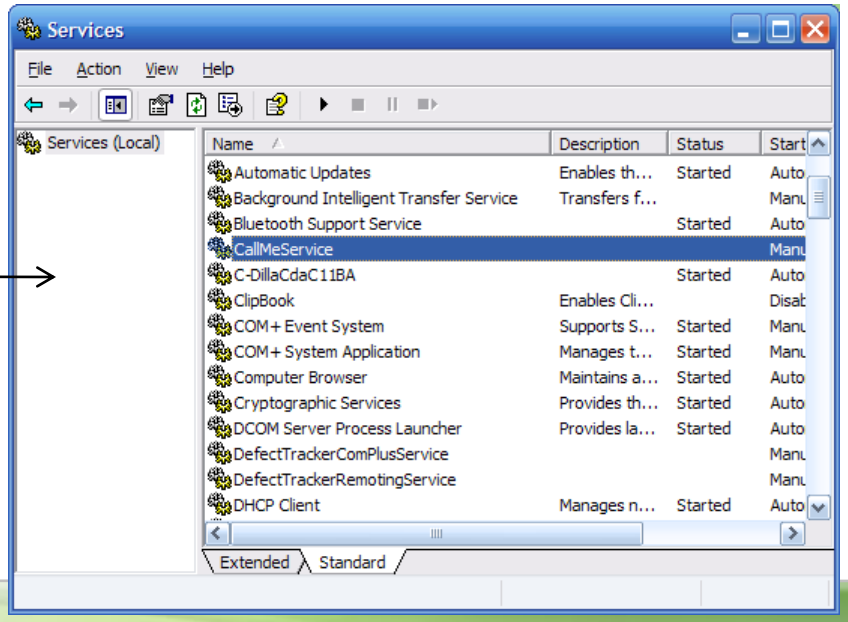
- **With self-hosting, you take on several key responsibilities**
  - These responsibilities can become costly to implement
  - However, you may not care about any/all of them
  - But if you do, look into a managed hosting solution



Hosting WCF in Windows Services

- **WCF services can be hosted within Windows Services**
  - A managed hosting solution built into Windows
- **Provides several key advantages**
  - Startup options: auto-start, manual commands (start & stop)
  - Doesn't constrain communication options
  - Choice of security identity
  - Management tool

Service Control Manager (SCM)



Implementing a Windows Service

```
public partial class InvoiceServiceHost : ServiceBase
{
    ServiceHost serviceHost =
        new ServiceHost(typeof(InvoiceService));

    protected override void OnStart(string[] args) {
        try {
            host.Open();
        }
        catch (Exception e) {
            ... // write exception details to event log
        }
    }
    protected override void OnStop() {
        if (null == serviceHost)
            return;
        if (serviceHost.State == CommunicationState.Faulted)
            serviceHost.Abort();
        else
            serviceHost.Close();
    }
}
```

derive from ServiceBase

create ServiceHost

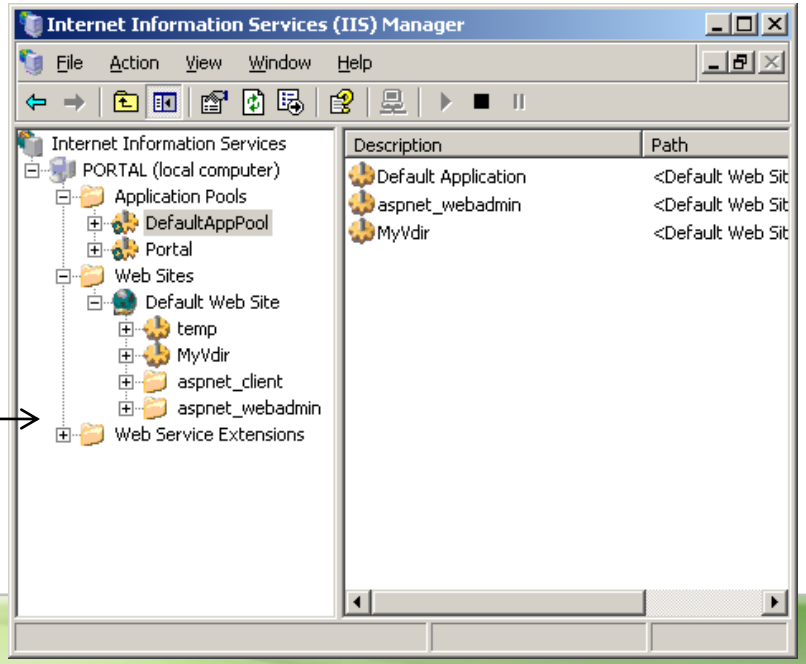
override OnStart

override OnStop

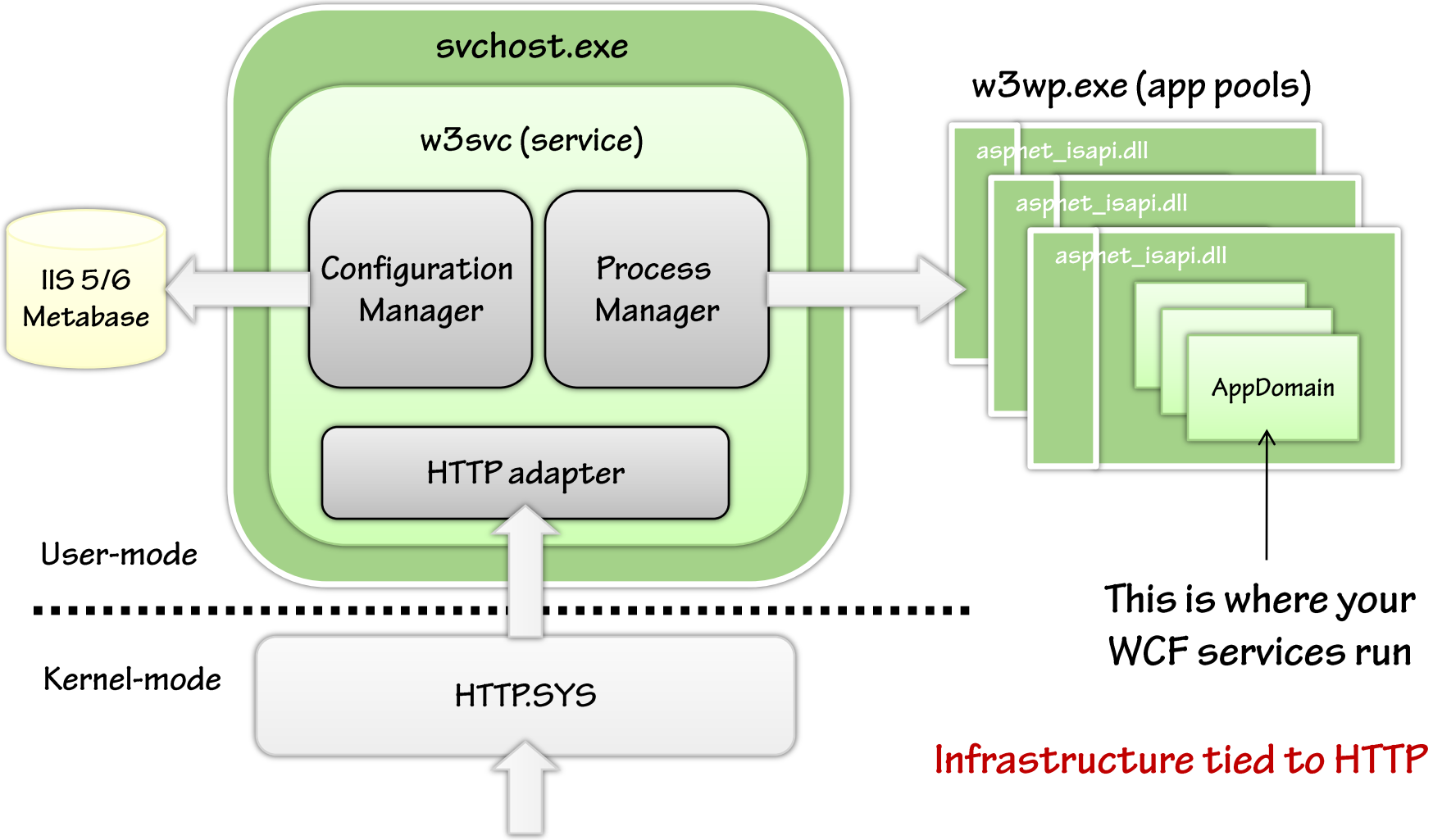
Hosting WCF in IIS 5/6

- **WCF services can be hosted in IIS 5/6 within ASP.NET applications**
  - A managed Web-hosting solution
- **Provides several key advantages**
  - Startup options: on-demand activation
  - Pooling, recycling, health monitoring
  - Choice of security identity
  - Management tool
- **One key disadvantage**
  - Restricts you to HTTP endpoints

Internet Information Services (IIS) Manager



IIS 6.0 architecture



Infrastructure tied to HTTP

Integrating WCF with IIS

- **You map incoming requests to a WCF ServiceHost using a .svc file**
  - Use the ASP.NET ServiceHost directive and the Service attribute
  - ASP.NET intercepts incoming requests and creates ServiceHost instance
  - Referenced assemblies must be in app's bin directory or the GAC
  - You can also place code in App_Code or inline (compiled on 1st request)

references class in assembly

invoiceservice.svc

```
<%@ ServiceHost Service="InvoiceServiceLibrary.InvoiceService" %>
```

references class defined in App_Code

```
<%@ ServiceHost Language="C#" Debug="true"
    Service="InvoiceService"
    CodeBehind="~/App_Code/InvoiceService.cs" %>
```

Configuring .svc services

- **You configure .svc services in the application's web.config**
  - The HTTP base address is set to the base address of the IIS application
  - WCF will ignore any base addresses specified in web.config

Web.config

```
<configuration>
  <system.serviceModel>
    <services>
      <service name="InvoiceServiceLibrary.InvoiceService">
        <endpoint
          address=""
          binding="basicHttpBinding"
          contract="InvoiceServiceLibrary.IInvoiceService"/>
      </service>
      ...
```

Typical to only use relative addresses

ServiceHostFactory

- **You can intercept ServiceHost creation using a ServiceHostFactory**
  - Derive from ServiceHostFactory and override CreateServiceHost
  - Specify your factory class in .svc using the Factory attribute

```
<%@ ServiceHost Factory="CustomServiceHostFactory"
                Service="InvoiceServiceLibrary.InvoiceService" %>
```

```
public class CustomServiceHostFactory : ServiceHostFactory
{
    protected override ServiceHost CreateServiceHost(
        Type serviceType, Uri[] baseAddresses)
    {
        return new CustomServiceHost(serviceType, baseAddresses);
    }
}
```

ASP.NET compatibility mode

- **In general, WCF services can run in one of two modes**
  - Mixed transports mode (default)
  - ASP.NET compatibility mode
- **With mixed transports mode**
  - You will not have access to any ASP.NET features (HttpContext, authorization, session state, etc)
- **With the ASP.NET compatibility mode**
  - You have access to all ASP.NET features
  - Similar to ASMX services

Enabling ASP.NET compatibility

```
<configuration>
  <system.serviceModel>
    <serviceHostingEnvironment
      aspNetCompatibilityEnabled="true"/>
    <services>
      <service name="InvoiceServiceLibrary.InvoiceService">
        <endpoint address=""
          binding="basicHttpBinding"
          contract="InvoiceServiceLibrary.IInvoiceService"/>
      </service>
      ...
```

Application-level flag

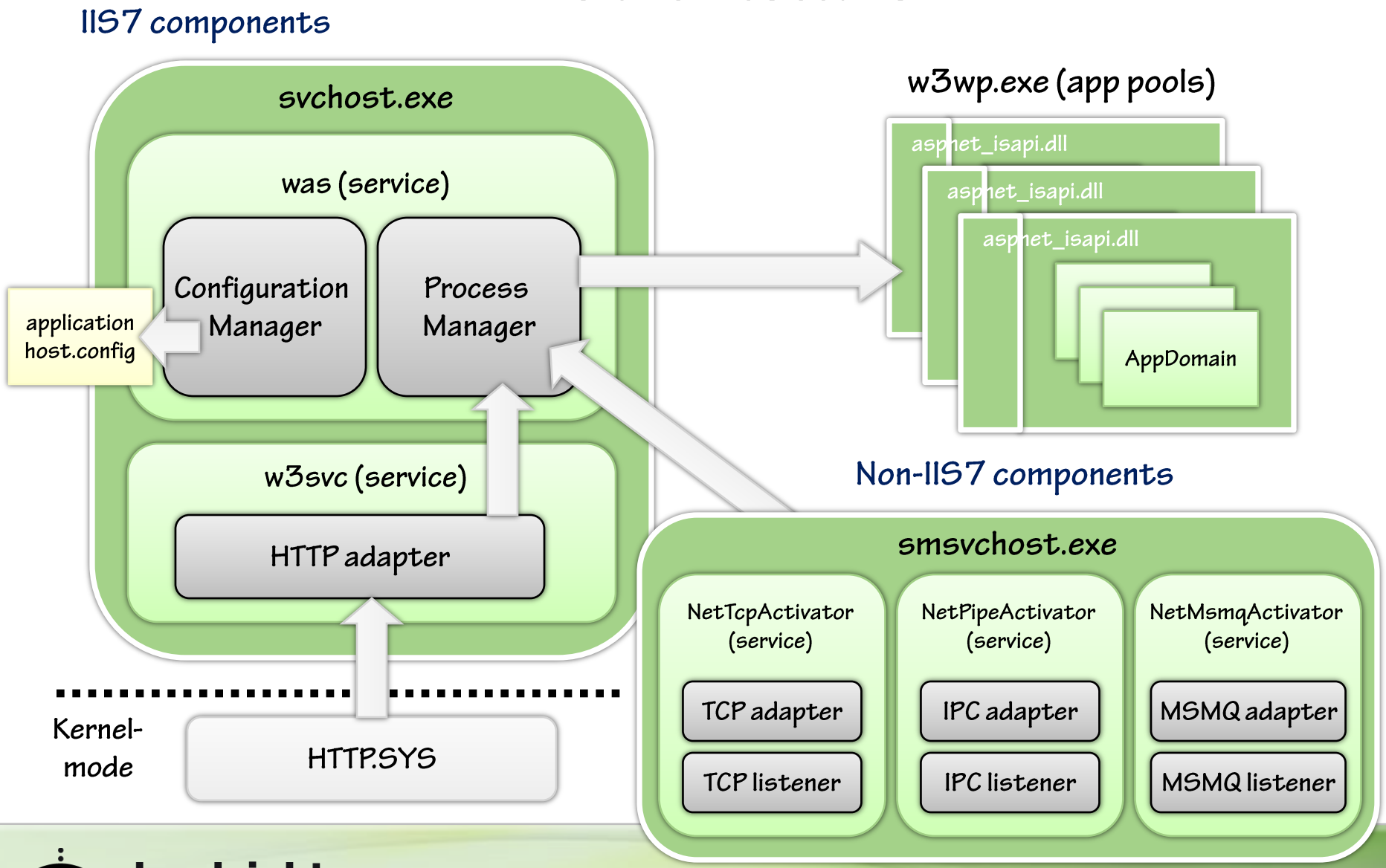
```
[AspNetCompatibilityRequirements(RequirementsMode=
   AspNetCompatibilityRequirementsMode.Required)]
public class InvoiceService : IInvoiceService
{
   ...
```

Require the host to provide ASP.NET compatibility

Windows Process Activation Services

- **Problem: WCF is transport-neutral but IIS 5/6 is tied to HTTP**
  - You can't take full advantage of WCF in this case
- **Solution: Windows Process Activation Service (WAS)**
  - Generalizes process activation & management
  - Allows WCF service activation over any transport
  - Part of IIS 7.0 but can be installed/configured separately
  - Ships with Windows Vista and Windows Server 2008

WAS architecture



Configuring WAS

- **Ensure you have WAS & the WCF activation components installed**
  - Windows Process Activation Service
  - Microsoft .NET 3.0 | WCF HTTP / Non-HTTP activation
- **Add configuration to applicationHost.config**
  - Add protocol bindings to Web sites
  - Enable protocols within applications
  - You can do this manually or via appcmd.exe
- **Configure endpoints like you normally would in web.config**
  - Now you can configure non-HTTP endpoints
- **Use .svc files just like before!**

Configuring WAS in applicationHost.config

applicationHost.config

Add protocol bindings to Web site

```
...
<bindings>
  <binding protocol="https" bindingInformation="*:443:" />
  <binding protocol="http" bindingInformation="*:80:" />
  <binding protocol="net.tcp" bindingInformation="808:*" />
  <binding protocol="net.pipe" bindingInformation="*" />
  <binding protocol="net.msmq" bindingInformation="localhost" />
  <binding protocol="msmq.formatname" bindingInformation="localhost" />
</bindings>
...
```

Enable protocols within application

```
...
<application path="/InvoiceService"
    enabledProtocols="http,net.tcp,net.pipe,net.msmq">
  <virtualDirectory path="/"
      physicalPath="C:\Service\InvoiceService" />
</application>
...
```

Configuring WAS with appcmd.exe

setupwas.cmd

Add protocol bindings to Web site

```
appcmd.exe set site "MySite" -bindings.[protocol='net.tcp',bindingInformation='808:*']
appcmd.exe set site "MySite" -bindings.[protocol='net.pipe',bindingInformation='*']
appcmd.exe set site "MySite" -bindings.[protocol='net.msmq',bindingInformation='localhost']

appcmd.exe set app "MySite/InvoiceService" /enabledProtocols:http,net.pipe,net.tcp,net.msmq
```

Enable protocols within application

Use "appcmd.exe /?" to get started

Summary

- **WCF services can be hosted in any .NET application**
- **ServiceHost provides the programming model**
  - Custom ServiceHost types are possible
- **Windows offers managed hosting environments**
  - Windows Services
  - IIS 5/6
  - IIS 7 & WAS
- **Choose the host that best fits your needs**

References

- **Hosting WCF Services**
  - http://www.code-magazine.com/article.aspx?quickid=0701041&page=1
- **Extend your WCF Services Beyond HTTP with WAS**
  - http://msdn.microsoft.com/msdnmag/issues/07/09/WAS/default.aspx
- **Pluralsight's WCF Wiki**
  - http://pluralsight.com/wiki/default.aspx/Aaron/WindowsCommunicationFoundationWiki.html