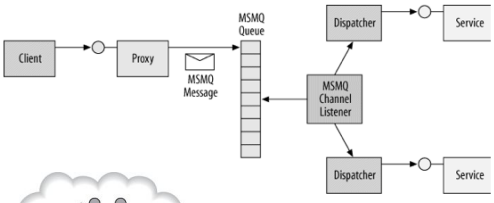
Queues and Service Bus

Dominic Duggan
Stevens Institute of Technology

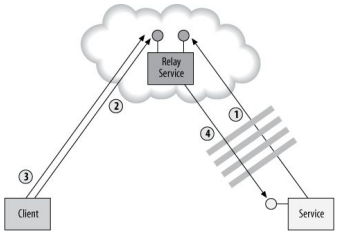
From Services to Service Bus

- Services

- Queued Services



- Service Bus Relay



WCF SECURITY

3

Authentication

- None
- Windows Domain
  - Kerberos
- Username and Password
- X509 Certificate
- Issued Token
  - E.g. Simple Web Token (SWT)

4

Message Transfer Security

- None
- Transport
  - HTTPS, TCP, IPC, MSMQ
  - Hardware acceleration
  - Point-to-point (intranet)
- Message
  - WS-Security
  - End-to-end (Internet)
- Mixed
  - Message for client credentials
  - Transport for message integrity and secrecy

5

Transfer Security Modes

Name	None	Transport	Message	Mixed	Both
BasicHttp Binding	Yes (default)	Yes	Yes	Yes	No
NetTcp Binding	Yes	**Yes (default)**	Yes	Yes	No
NetNamed Pipe Binding	Yes	Yes (default)	No	No	No
WSHttp Binding	Yes	Yes	**Yes (default)**	Yes	No
NetMsmq Binding	Yes	Yes (default)	Yes	No	Yes

6

Protection Level

- Switch for *intranet* Transport security
  - None
  - Sign
    - MSMQ default
  - Encrypt and Sign
    - Pick this

7

Constraining Message Protection

- Constrained, independently, by client & service:
  - At service contract
  - At operation contract
  - At fault contract
- Satisfied by Transport or Message security
  - Internet bindings

```
[ServiceContract(ProtectionLevel=ProtectionLevel.
EncryptAndSign)]
interface IMyContract
{…}
```

8

CLIENT AUTHENTICATION

9

Client Credentials

- How do clients authenticate to service?
  - Windows domain (Kerberos)
    - Default for intranet
    - Also for WSHttpBinding!
  - Shared secret (user name & password)
  - X509 certificate
  - Issued token
    - With Message security

10

Bindings and Transport security client credentials

Name	None	Windows	Username	Certificate
BasicHttp Binding	Yes (default)	Yes	Yes	Yes
NetTcp Binding	Yes	**Yes (default)**	No	Yes
NetNamed Pipe Binding	Yes	Yes (default)	No	No
WSHttp Binding	Yes	**Yes (default)**	Yes	Yes
NetMsmq Binding	Yes	Yes (default)	No	Yes

11

Bindings and Message security client credentials

Name	None	Windows	Username	Certificate	Issued Token
BasicHttp Binding	No	No	No	Yes	No
NetTcp Binding	Yes	**Yes (default)**	Yes	Yes	Yes
NetNamed Pipe Binding	N/A	N/A	N/A	N/A	N/A
WSHttp Binding	Yes	**Yes (default)**	Yes	Yes	Yes
NetMsmq Binding	Yes	Yes (default)	Yes	Yes	Yes

12

Identity Management

- Which security identity is sent by client?
- Which security identity does service execute under?
  - Own identity
  - Impersonate client
  - Some combination

13

Identity Management

- .NET: `IIdentity` interface:

```
public interface IIdentity {
  string AuthenticationType {get;}
  bool IsAuthenticated {get;}
  string Name {get;}
}
```

- Implementations:
  - `WindowsIdentity`
  - `GenericIdentity`
  - `X509Identity`

14

Security Call Context

- Available as `ServiceSecurityContext.Current`
- Also available from `OperationContext`

```
class ServiceSecurityContext {
  public static ServiceSecurityContext
                Current {get;}
  public bool IsAnonymous {get;}
  public IIdentity PrimaryIdentity {get;}
  public WindowsIdentity WindowsIdentity {get;}
}
```

15

Security Principal

- Grouping identity and role membership

```
public interface IPrincipal {
  IIdentity Identity {get;}
  bool IsInRole(string role);
}
```

- Available from TLS:

```
IPrincipal principal = Thread.CurrentPrincipal;
string userName = principal.Identity.Name;
bool isAuthenticated =
        principal.Identity.IsAuthenticated;
```

16

CLIENT AUTHORIZATION

17

Requiring Permissions

```
[ServiceContract]
interface IMyContract {
  [OperationContract]
  void MyMethod();
}

class MyService : IMyContract {
  [PrincipalPermission
      (SecurityAction.Demand,Role="Manager")]
  public void MyMethod() {...}
}
```

18

Authorization Mode

- ServiceHostBase class supports:
  - None
  - UseWindowsGroups
  - UseAspNetRoles
  - Custom

19

Authorization Mode

- Define as a service behavior:

```
<services>
  <service behaviorConfiguration = "AspNetRoles">
    ...
  </service>
</services>
<behaviors>
  <serviceBehaviors>
    <behavior name = "AspNetRoles">
      <serviceAuthorization
        principalPermissionMode="UseAspNetRoles"/>
    </behavior>
  </serviceBehaviors>
</behaviors>
```

20

Security Auditing

```
<services>
  <service behaviorConfiguration = "MySecurityAudit">
    ...
  </service>
</services>
<behaviors>
  <serviceBehaviors>
    <behavior name = "MySecurityAudit">
      <serviceSecurityAudit
        auditLogLocation = "Default"
        serviceAuthorizationAuditLevel =
             "SuccessOrFailure"
        messageAuthenticationAuditLevel =
             "SuccessOrFailure"/>
    </behavior>
  </serviceBehaviors>
</behaviors>
```

Default
Application
Security

None
Success
Failure
SuccessOrFailure

21

SECURITY SCENARIOS

22

Intranet Scenario

- Bindings: NetTcp, NetNamedPipe, NetMsmq
- Transfer security: Transport
- Client credentials: Windows (or UserName)
- Protection level: Encrypt and Sign

23

Internet Scenario

- Binding: WSHttp (no security with BasicHttp)
- Transfer security: Message
- Client credentials: UserName
- Protection level: Encrypt and Sign
  - Client generates private session key
  - Passed encrypted with service public key

24

Internet Scenario

- Identifying service certificate:
  - Hard-code in config file, or
  - Store certs in trust store, negotiate with service
- Validating service certificate:
  - Peer trust
  - Chain trust (only use private CA)
  - Custom

25

Internet Scenario

- Validating service cert:

```
<client>
  <endpoint
    behaviorConfiguration = "ServiceCert"
  </endpoint>
</client>
<endpointBehaviors>
  <behavior name = "ServiceCert">
    <clientCredentials>
      <serviceCertificate>
        <authentication
           certificateValidationMode = "PeerTrust"/>
      </serviceCertificate>
    </clientCredentials>
  </behavior>
</endpointBehavior>
```

26

Internet Scenario

- Test Certificate
  - Default: service cert name matches service host domain
  - Otherwise explicitly specify test cert name:

```
<client>
  <endpoint
    address  = "http://localhost:8001/MyService"
    binding  = "wsHttpBinding"
    contract = "IMyContract"
    <identity>
      <dns value = "MyServiceCert"/>
    </identity>
  </endpoint>
</client>
```

  - Also necessary with Service Bus Relay (later)

27

B2B Scenario

- Scenario:
  - Clients identify themselves using X509 certs
  - Rely on HTTP for transport
  - Multiple intermediaries possible
- Approach:
  - Use Message transfer security mode
  - Message protected by service-side cert
  - Clients authenticate with certificates
  - Validate client certs using *peer trust*

28

B2B Scenario

- Use Message transfer security

```
<bindings>
  <wsHttpBinding>
    <binding name = "WSCertificateSecurity">
      <security mode = "Message">
        <message
          clientCredentialType = "Certificate" />
      </security>
    </binding>
  </wshttpBinding>
</binding>
```

None
Windows
UserName
Certificate
IssuedToken

29

B2B Scenario

- Validate clients using peer trust

```
<services> <service behaviorConfiguration = "B2B" ...>...
</services>
<behaviors>
  <serviceBehaviors>
    <behavior name = "B2B">
      <serviceCredentials>
        <serviceCertificate ... />
        <clientCertificate>
          <authentication
            certificateValidationMode = "PeerTrust"/>
        </clientCertificate>
      </serviceCredentials>
    <behavior>
  <serviceBehaviors>
</behaviors>
```

ChainTrust
PeerTrust
None
Custom

30

B2B Scenario

- Set certificate in the client

```
<client> <endpoint behaviorConfiguration = “B2B” ...>...
<behaviors>
  <endpointBehaviors>
    <behavior name = “B2B”>
      <clientCredentials>
        <clientCertificate
           findValue     = “MyClientCert”
           storeLocation = “LocalMachine”
           storeName     = “My”
           x509FindType  = “FindBySubjectName”
        />
      </clientCredentials>
    <behavior>
  <endpointBehaviors>
</behaviors>
```

31

B2B Scenario

- Hard-coding client cert in proxy

```
class MyContractClient: ClientBase <... >,... {
  public MyContractClient() {
    SetCertificate();
  }

  void SetCertificate() {
    ClientCredentials.ClientCertificate
     .SetCertificate(StoreLocation.LocalMachine,
                     StoreName.My,
                     X509FindType.FindBySubjectName,
                     "MyClientCert");
  }
  ...
}
```

32

B2B Scenario

- Client authorization scenario #1:
  - Generic principal (`PrincipalPermissionMode.None`)
  - Ex: only one client!
- Client authorization scenario #2:
  - Principal based on credentials (`PrincipalPermissionMode.UseAspNetRoles`)
  - Enable role provider
  - Add client cert & thumbprint to membership store
  - Assign roles to client cert

33

QUEUED SERVICES

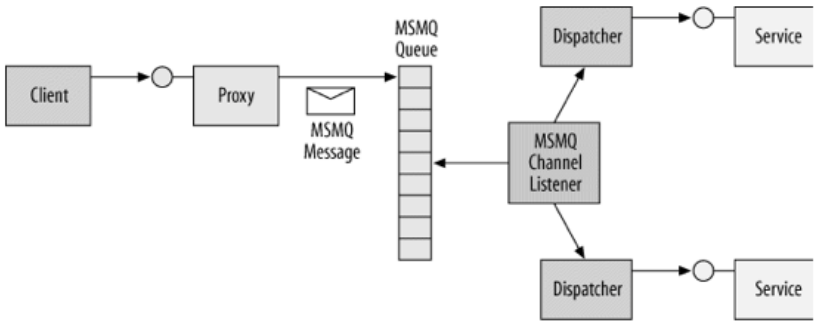
34

Queued Services

- Motivation:
  - Availability
    - Disconnected operation
  - Disjoint work
    - Business workflows
  - Compensating work
    - Initiating transaction
    - Completion transaction
  - Load leveling

35

Queued Calls Architecture



36

Contracts

- Service:

```
[ServiceContract]
interface IMyContract {
  [OperationContract(IsOneWay = true)]
  void MyMethod();
}
class MyService : IMyContract {
  public void MyMethod() { ... }
}
```

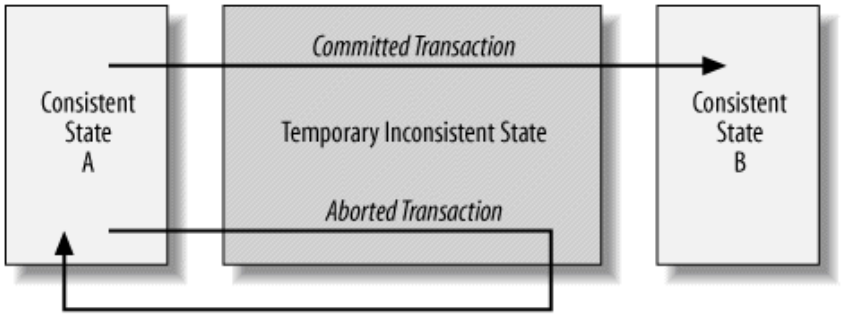
- Client:

```
MyContractClient proxy = new MyContractClient();
proxy.MyMethod();
proxy.Close();
```

37

Transactions

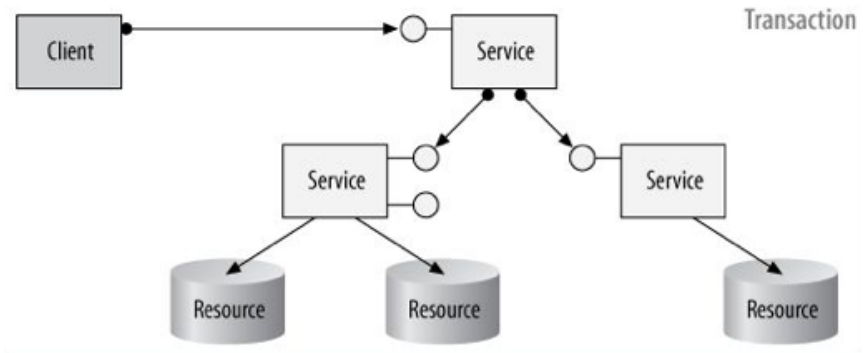
cf CS548



38

Distributed Transactions

cf CS549

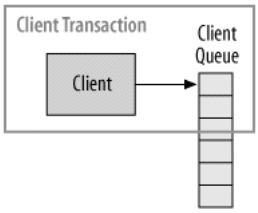


39

Queues and Transactions

cf CS548

- Transactional queue
  - Persistent
  - Message posting and removal transactional
  - New transaction if no ambient



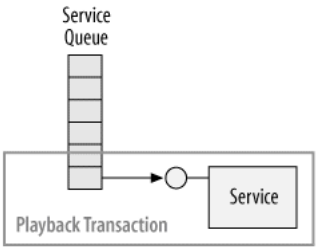
40

Delivery and Playback

cf CS548

- Delivery transaction
  - "guaranteed delivery"
  - Failure notification o/w

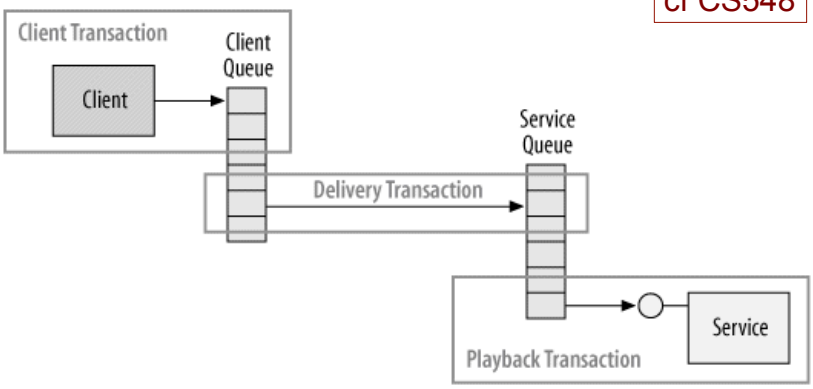
- Playback transaction
  - Auto-retry
  - Queued call kept short
  - Not long-lived async



41

Queued Service Transactions

cf CS548



42

Playback Transaction

cf CS548

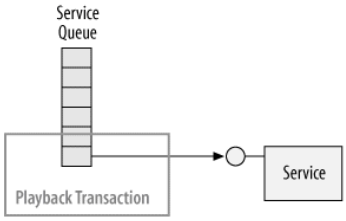
```
[ServiceContract]
interface IMyContract {
  [Operation(IsOneWay = true)]
  void MyMethod();
}
class MyService : IMyContract {
  [OperationBehavior
   (TransactionScopeRequired = true)]
  public void MyMethod() {
    Transaction transaction = Transaction.Current;
    transaction.Abort();
  }
}
```

43

Ignoring Playback Transaction

cf CS548

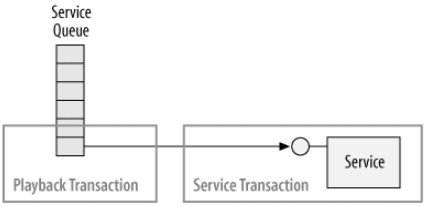
```
[ServiceContract]
interface IMyContract {
  [Operation(IsOneWay = true)]
  void MyMethod();
}
class MyService : IMyContract {
  public void MyMethod() {
    Transaction transaction = Transaction.Current;
    Debug.Assert(transaction==null);
  }
}
```



Using Separate Transaction

cf CS548

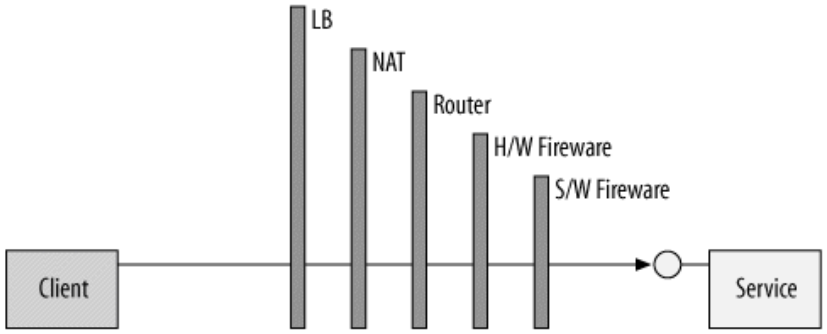
```
[ServiceContract]
interface IMyContract {
  [Operation(IsOneWay = true)]
  void MyMethod();
}
class MyService : IMyContract {
  public void MyMethod() {
    using (TransactionScope scope =
           new TransactionScope()) {
      ...
      scope.Complete();
    }
  }
}
```



SERVICE BUS

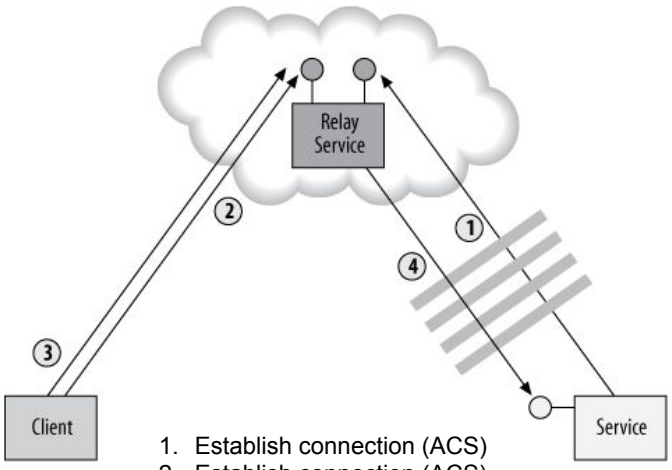
46

Internet Connectivity Challenges



47

Relay Service



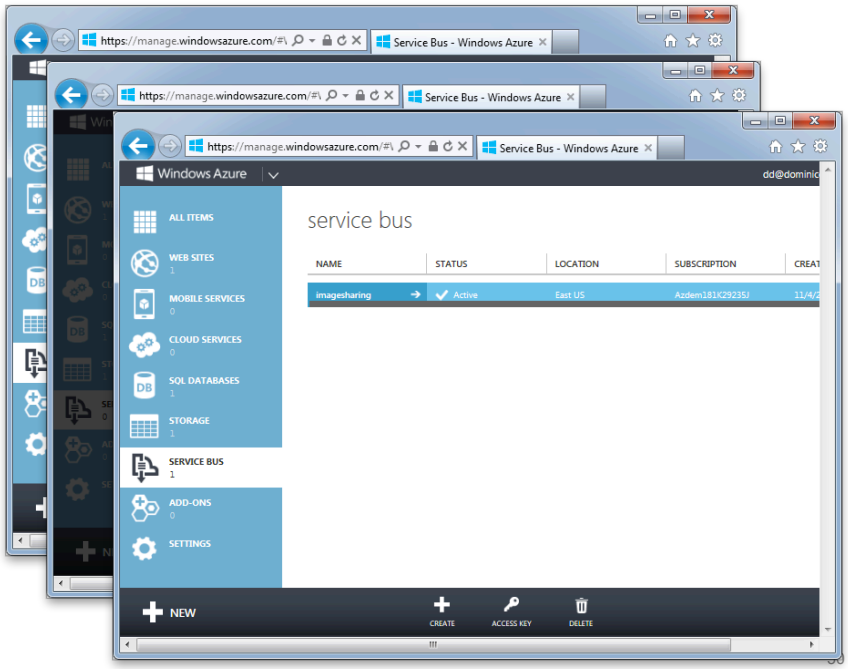
1. Establish connection (ACS)
2. Establish connection (ACS)
3. Client requests service
4. Call forwarded

48

Service Bus as Relay Service

- Service Bus: Effectively DMZ in the cloud
- Access Control Service (ACS)
  - Off-loading authentication & security mangt
- Hosting
  - Service authenticates to relay service
  - Self-hosting
  - WAS: deploy running App Server AppFabric configured for auto-start

49



SERVICE BUS ENDPOINTS

51

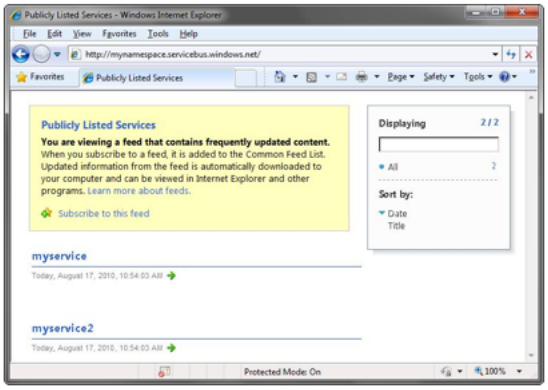
Relay Service Address

- Defines service namespace:
  `[base-address]/[optional-uri]/…/[optional-uri]`
- Where [base-address] has the form
  `[scheme]://[service-namespace].[service-bus-address]`
- Examples
  `sb://MyNamespace.servicebus.windows.net`
  `sb://MyNamespace.servicebus.windows.net`
  `/MyService/MyEndpoint`
  `http://MyNamespace.servicebus.windows.net`
  `https://MyNamespace.servicebus.windows.net`

52

Service Bus Registry

- Service bus ATOM feed of listening services



53

Service Bus Registry

- Services must explicitly publish to registry:

```
IEndpointBehavior registryBehavior =
  new ServiceRegistrySetting(DiscoveryType.Public);

ServiceHost host =
  new ServiceHost(typeof(MyService));

foreach (ServiceEndpoint endpoint in
             host.Description.Endpoints) {
  endpoint.Behaviors.Add(registryBehavior);
}

host.Open();
```

54

Service Bus Bindings

- TCP Relay Binding
  - `NetTcpRelayBinding`
- Oneway Relay Binding
  - `NetOnewayRelayBinding`
- Event Relay Binding
  - NetEventRelayBinding
- WS Relay Binding
  - `WS2007HttpRelayBinding`
  - Slow; for interoperability

55

TCP Relay Binding

- Endpoint declaration:

```
<endpoint
   address =
     "sb://MyNamespace.servicebus.windows.net/…"
   binding = "netTcpRelayBinding"
   contract = "…"
/>
```

- Best performance and throughput
- Supports request-reply, one-way, *and duplex*
- Always uses transport session
- Not interoperable

56

Client Binding

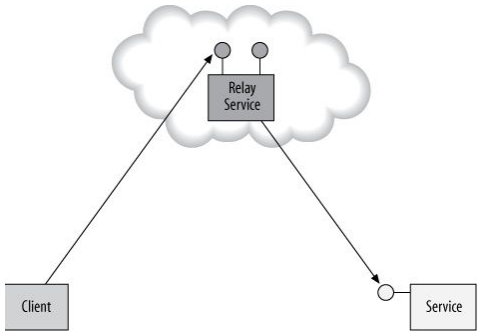
- Client endpoint:

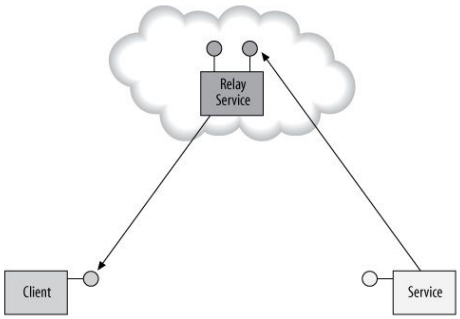
```
<client>
   <endpoint
      name = "myClientEndpoint"
      contract = "IMyContract"
      address =
        "sb://MyNamespace.servicebus.windows.net/…"
      binding = "netTcpRelayBinding"
      behaviorConfiguration = "…"
   />
</client>
```

57

Connection Modes

- TCP Relayed Mode
  - Default mode
  - Supports duplex MEP

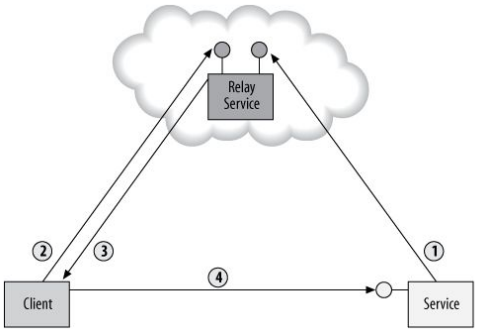




58

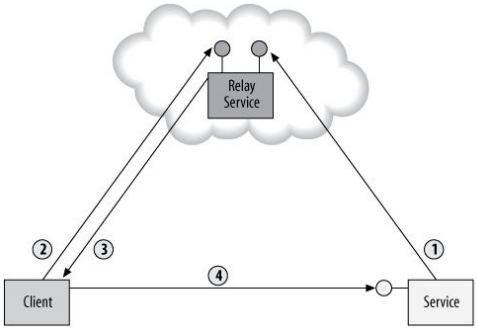
Connection Modes

- TCP Hybrid Mode
  - Promote from relay



59

- TCP Hybrid Mode




```
<endpoint
  address = "sb://MyNamespace.service.windows.net..."
  binding = "netTcpRelayBinding"
  bindingConfiguration = "Hybrid"
  contract = "..." />
<bindings
  <netTcpRelayBinding
    <binding name = "Hybrid"
             connectionMode = "Hybrid" ... />
  </netTcpRelayBinding>
</bindings>
```

60

Service Bus Bindings

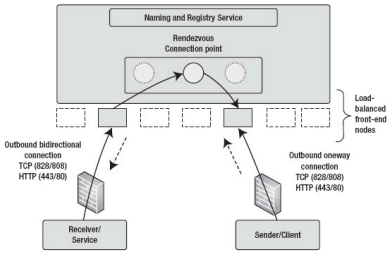
- WS 2007 Relay Binding

```
<endpoint
  address =
    "http:// MyNamespace.servicebus.windows.net/..."
  binding = "ws2007HttpRelayBinding"
  contract = "…" />
```

61

Service Bus Bindings

- One-Way Relay Binding
  - One receiver, many senders
  - Not fire-and-forget
    - Service must be running
    - Transport session between client and relay
    - Communication error ⇒ new client proxy
  - Client isolated from service failures

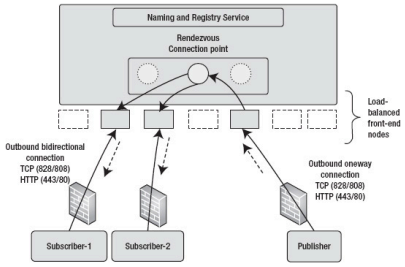


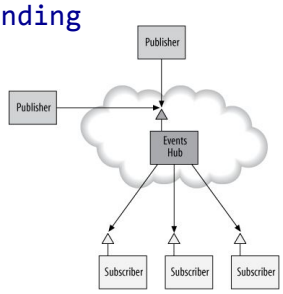
62

Service Bus Bindings

- Event Relay Binding
  - Many receivers, many senders
  - Client side:

    `NetEventRelayBinding : NetOnewayRelayBinding`
  - Event publishing





SERVICE BUS AUTHENTICATION

Service Bus Authentication

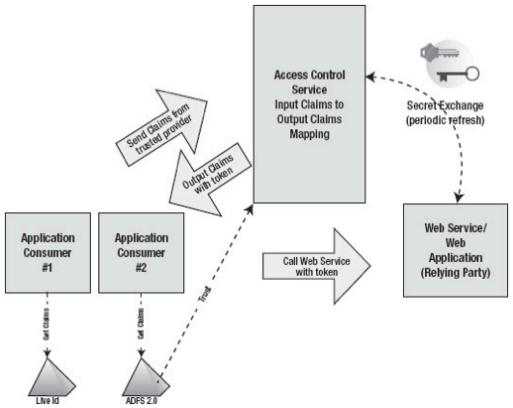
- Service required to authenticate to relay
- Client may or may not authenticate
- Client and service:
  - Security tokens issued by ACS
  - Claims-based identity model

65

Access Control Service (ACS)

cf CS594

- Claims-based model:
  - ACS and IdP (e.g. LiveID) trust each other
  - ACS and RP (SB or service) trust each other (signing key)
  - ACS maps client input claims to output claims
  - Client sends output claim (token) to service

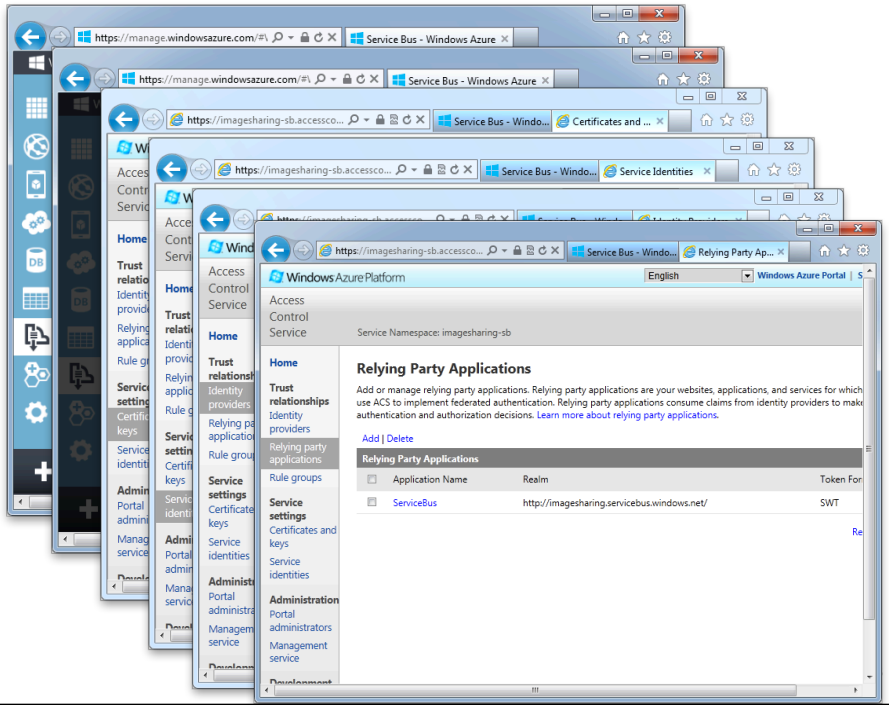


Service Bus and ACS

cf CS594

- SB namespace: `MyNamespace`
- ACS namespace: `MyNamespace-sb`
  - Realm: `http://MyNamespace.service.windows.net`
  - Token type (output claims): SWT
  - Rules map input claims to SB permission claims
    - `Send`
    - `Listen`
    - `Manage`

67



Example SB Permissions

cf CS594

Operation	Claim Required	Claim Scope
Listen on SB relay	Listen	*Any* service namespace address
Send on SB relay	Send	*Any* service namespace address
Create queue	Manage	*Any* service namespace address
Send message to queue	Send	*Any* valid queue address
Receive messages from queue	Listen	*Any* valid queue address

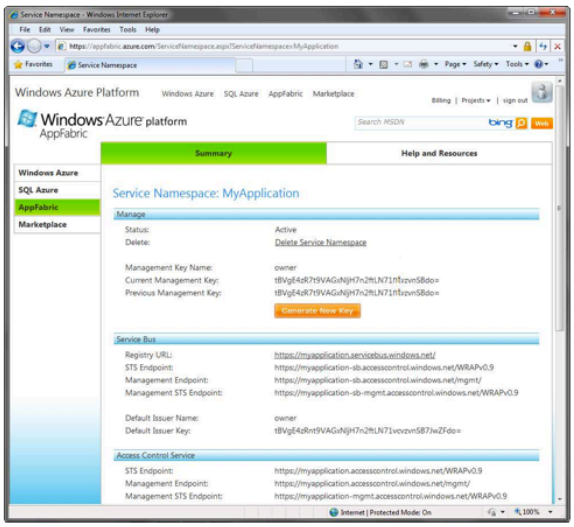
69

Token Provider

- API: input claims → ACS permissions
  - **Shared secret**
    - Service identity & key in SB
  - Simple Web Token (SWT)
    - WS-Federation
  - SAML
    - WS-Federation
  - Others (e.g. OAuth)
- NB client of *service bus*

70

Configuring Secret Keys



71

Clients of Token Providers

- `TransportClientEndpointBehavior`
  - Service bus credentials
- `NamespaceManager`
  - Manage queues, topics and subscriptions
- `MessagingFactory`
  - Communication via queues, topics and subscriptions

72

Service Bus Authentication

- Configure auth at the endpoints:

```
public class TransportClientEndPointBehavior :
                                IEndpointBehavior {
  ~~TransportClientCredentials Credentials {get;}~~
  ~~TransportClientCredentialType~~
        ~~CredentialType {get; set;}~~
  TokenProvider TokenProvider {get; set;}
}
```

73

Service: Providing Credentials

```
TransportClientEndpointBehavior credentials =
  new TransportClientEndpointBehavior();
credentials.TokenProvider =
  TokenProvider
    .CreateSharedSecretTokenProvider
                  (ISSUER-NAME, ISSUER-SECRET);

Servicehost host = new ServiceHost(typeof(MyService));

foreach (ServiceEndpoint endpoint in
             host.Description.Endpoints) {
  endpoint.Behaviors.Add(credentials);
}
host.Open();
```

74

Client: Providing Credentials

```
TransportClientEndpointBehavior credentials =
  new TransportClientEndpointBehavior();
credentials.TokenProvider =
  TokenProvider
    .CreateSharedSecretTokenProvider
                    (ISSUER-NAME, ISSUER-SECRET);

MyContractClient proxy = new MyContractClient();
proxy.Endpoint.Behaviors.Add(credentials);

proxy.MyMethod();
proxy.Close();
```

75

Credentials in Config File

```
<endpoint behaviorConfiguration = "SharedSecret"
  ...  />
...
<behaviors>
  <endpointBehaviors>
    <behavior name = "SharedSecret">
      <transportClientEndpointBehavior>
        <tokenProvider>
          <sharedSecret issuerName = ISSUER-NAME
                        issuerSecret = ISSUER-SECRET
        </tokenProvider>
      </transportClientEndpointBehavior>
    </behavior>
  </endpointBehavior>
</behaviors>
```

76

Unauthenticated Clients

- Allow unauthenticated client access
  - …to service endpoint
- Service must authenticate client
  - More exposure to risk
  - Use Message security to xfer client credentials

77

Unauthenticated Clients

```
public class NetTcpRelayBinding {
  NetTcpRelaySecurity Security {get;} ...
}
public class NetTcpRelaySecurity {
  RelayClientAuthenticationType
      RelayClientAuthenticationType {get; set;}
 // Values: None, RelayAccessToken (default)

  EndToEndSecurityMode Mode {get; set;}
  TcpRelayTransportSecurity
      Transport {get;}
  // Values: None, Sign, EncryptAndSign
  MessageSecurityOverlayRelayConnection
      Message {get;}
  // Credential type: None, Windows (default),
  // UserName, Certificate, IssuedToken
}
```

78

Unauthenticated Clients

- Configuring host to allow unauthenticated clients

```
<services>
  <service ...>
    <endpoint
      binding = "netTcpRelayBinding"
      bindingConfiguration = "NoServiceBusAuth" ... />
  </service>
</services>
<bindings>
  <netTcpRelayBinding>
    <binding name = "NoServiceBusAuth">
      <security relayClientAuthenticationType = "None" />
    </binding>
  </netTcpRelayBinding>
</bindings>
```

RelayAccessToken
None

79

Unauthenticated Clients

- Configuring client for unauthenticated access

```
<client>
  <endpoint
      binding = "netTcpRelayBinding"
      bindingConfiguration = "NoServiceBusAuth" ... />
</client>
...
    <binding name = "NoServiceBusAuth">
      <security relayClientAuthenticationType = "None" />
    </binding>
...
```

80

SERVICE BUS TRANSFER SECURITY

81

Transfer Security

- Service bus: “end-to-end” security
  - None
  - Transport
  - Message
  - Transport with message credential

82

Transfer ("End-to-End") Security

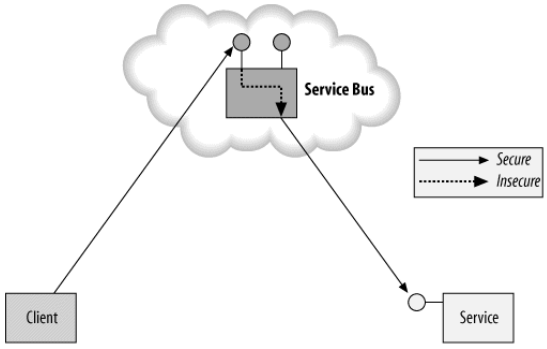
```
public class NetTcpRelayBinding {
  NetTcpRelaySecurity Security {get;} ...
}
public class NetTcpRelaySecurity {
  RelayClientAuthenticationType
      RelayClientAuthenticationType {get; set;}
  // Values: None, RelayAccessToken (default)

  EndToEndSecurityMode Mode {get; set;}
  TcpRelayTransportSecurity
      Transport {get;}
  // Values: None, Sign, EncryptAndSign
  MessageSecurityOverlayRelayConnection
      Message {get;}
  // Credential type: None, Windows (default),
  // UserName, Certificate, IssuedToken
}
```

83

Transport Security

- SSL, HTTPS
- Client calls always anonymous
- Not end-to-end



84

Message Security

- WS-Security, XML-Encryption
- Encrypt body using service-provided cert
- Client credentials for service authorization
- Claims-based authorization

85

TCP Relay Binding

- TCP Relay Binding: default Transport security
- `ServiceSecurityContext` is null (anonymous)
- Protection level for messages:
  - `NetTcpRelayBindingBase.Security .Transport.ProtectionLevel`
  - Default: `ProtectionLevel.EncryptAndSign`

86

Anonymous Message Security

- No client credentials in message
- Client must still validate service certificate
  - Peer Trust: client "Trusted People" folder
- Endpoint identity: based on service cert

- Set host `PrincipalPermissionMode` to `None`
  - Same principal as with Transport security

87

Anonymous Message Security

- Service Side: endpoint configuration

```
<service name = "..."
  behaviorConfiguration = "MessageSecurity">
  <endpoint ...
    binding = "NetTcpRelayBinding"
    bindingConfiguration = "MessageSecurity" />
</service>
...
```

88

Anonymous Message Security

- Service Side: Specify server certificate

```
<serviceBehaviors>
   <behavior name = "MessageSecurity">
     <serviceCredentials>
       <serviceCertificate
         findValue     = "MyServiceCert"
         storeLocation = "LocalMachine"
         storeName     = "My"
         x509FindType  = "FindBySubjectName" />
     </serviceCredentials>
     <serviceAuthorization
          principalPermissionMode = "None" />
   </behavior>
</serviceBehaviors>
```

UseWindowGroups
UseAspNetRoles
None
Custom

89

Anonymous Message Security

- Service Side: no client credentials

```
<bindings>
  <netTcpRelayBindings>
    <binding name = "MessageSecurity">
      <security mode = "Message">
        <message clientCredentialType = "None" />
      </security>
    </binding>
  </netTcpRelayBinding>
</bindings>
```

None
Windows
UserName
Certificate
IssuedToken

90

Anonymous Message Security

- Configuring client: endpoint identity based on cert

```
<client>
  <endpoint
    behaviorConfiguration = "ServiceCertificate"
    binding = "NetTcpRelayBinding"
    bindingConfiguration = "MessageSecurity" >
    <identity>
      <dns value = "MyServiceCert"/>
    </identity>
    ...
  </endpoint>
</client>
```

91

Anonymous Message Security

- Configuring client: validate service certificate

```
<behaviors>
  <endpointBehaviors>
    <behavior name = "ServiceCertificate">
     <clientCredentials>
       <serviceCertificate>
         <authenticate
           certificateValidationMode = "PeerTrust"/>
       </serviceCertificate>
     </clientCredentials>
   </behavior>
  </endpointBehaviors>
<behaviors>
```

ChainTrust
PeerTrust
None
Custom

92

Anonymous Message Security

- Configuring client: no credentials to service

```
<bindings>
  <netTcpRelayBindings>
    <binding name = “MessageSecurity”>
      <security mode = “Message”>
        <message clientCredentialType = “None” />
      </security>
    </binding>
  </netTcpRelayBinding>
</bindings>
```

None
Windows
UserName
Certificate
IssuedToken

93

Message Security with Credentials

- Configuring client: authenticate based on username

```
<bindings>
  <netTcpRelayBindings>
    <binding name = “MessageSecurity”>
      <security mode = “Message”>
        <message clientCredentialType = “UserName” />
      </security>
    </binding>
  </netTcpRelayBinding>
</bindings>
```

None
Windows
UserName
Certificate
IssuedToken

94

Message Security with Credentials

- Configuring client for authentication at service

```
MyContractClient proxy = new MyContractClient();

proxy.ClientCredentials.UserName.UserName =
    "MyUserName";
proxy.ClientCredentials.UserName.Password =
    "MyPassword";

proxy.MyMethod();

proxy.Close();
```

95

Message Security with Credentials

- Configuring service: how to authenticate client

```
<behaviors>
  <serviceBehaviors>
     <behavior name = "MessageSecurity">
       <serviceCredentials>
         <userNameAuthentication
           userNamePasswordValidationMode =
                          "MembershipProvider" />
       </serviceCredentials>
       <serviceAuthorization
            principalPermissionMode = "UseAspNetRoles" />
     </behavior>
  </serviceBehaviors>
<behaviors>
```

Windows
Membership Provider
Custom

UseWindowsGroups
UseAspNetRoles
None
Custom

96

Mixed Security

- Configuring service or client
- Default: auth based on Windows credentials

```
<bindings>
  <netTcpRelayBinding>
    <binding name = "MixedSecurity">
      <security mode="TransportWithMessageCredential">
        <message ClientCredentialType = "UserName"/>
      </security>
    </binding>
  </netTcpRelayBinding>
</bindings>
```

None
Windows
UserName
Certificate
IssuedToken

97

WS Relay Binding

- Binding: specify `https` (Transport security)
- Anonymous client
- Message/Mixed: Use https to protect ACS token

```
<endpoint
  address =
      "https://MyNamespace.servicebus.windows.net/..."
      binding = "ws2007HttpRelayBinding"
      ...
/>
```

98

One-Way Relay Binding

- Default: Transport security (`EncryptAndSign`)
- Mixed not supported
- Cannot negotiate service certificate (must specify)
- Security call context: `CN=servicebus.windows.net`

```
<client>
  <endpoint behaviorConfiguration = "ServiceCert"
    ...
  </endpoint>
</client>
```

99

One-Way Relay Binding

- Client Side: Specify service cert

```
<endpointBehaviors>
   <behavior name = "ServiceCert">
     <clientCredentials>
       <serviceCertificate>
         <scopedCertificates>
           <add targetUri="sb://MyNamespace.servicebus..."
             findValue     = "MyServiceCert"
             storeLocation = "LocalMachine"
             storeName     = "My"
             x509FindType  = "FindBySubjectName" />
         </scopedCertificates>
       </serviceCertificate>
     </clientCredentials>
   </behavior>
</endpointBehaviors>
```

100

Binding & Transfer Security

Binding	None	Transport	Message	Mixed
TCP (relayed)	Yes	Yes (default)	Yes	Yes
TCP (hybrid)	Yes	No	Yes	No
WS	Yes	Yes (default)	Yes	Yes
One-way	Yes	Yes (default)	Yes	Yes

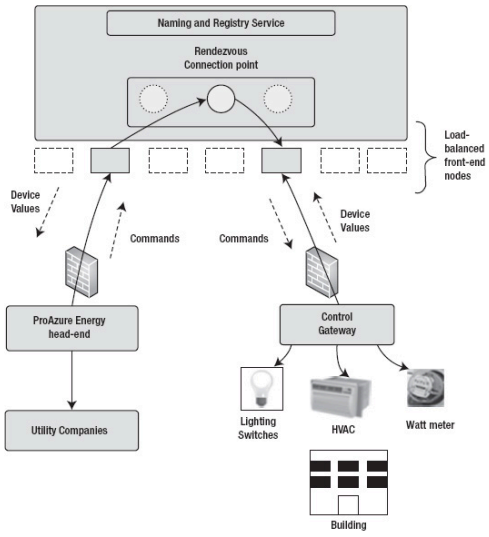
101

EXAMPLE: ENERGY MANAGEMENT SERVICE

102

Energy Service Architecture

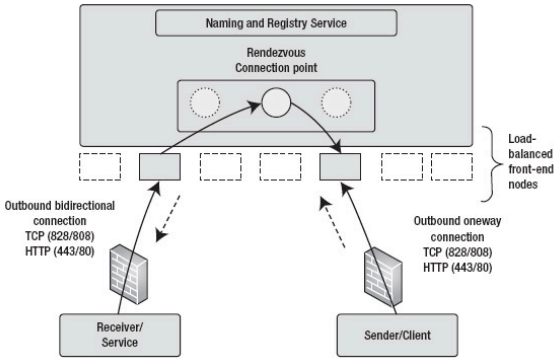
- Households
  - Devices (thermostat, etc)
  - *Control Gateways*
  - Event notifications
- Utilities
  - *Head end servers*
  - Collect notifications
  - Send commands



103

HVAC Control Notification Service

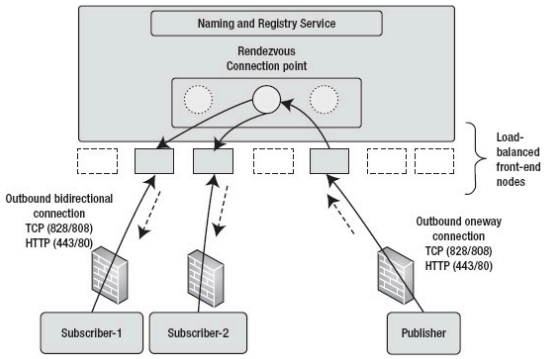
- Oneway Relay Binding
  - Control gateway → head-end server: event notify



104

Availability Notification Service

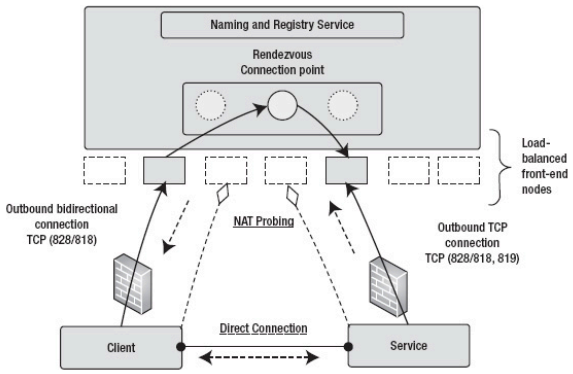
- Net Event Relay Binding
  - Control gateway → head-end server(s): availability



105

TCP Relay Binding

- Point-to-point bidirectional
  - Relayed
  - Direct
  - Hybrid
- Application:
  - Head-end server → Control gateway
  - Get and set operations on back-end devices



106

AZURE STORAGE QUEUES

107

Azure Storage Queues

- Max message size 64KB
- FIFO not guaranteed
- At-least-once semantics
- Received messages always Base64-encoded
- 7-day expiration period
- Based on HTTP/S

108

Azure Storage Queues

- Queue URI:
  `http://acct-name.queue.core.windows.net/qname`
- Constraints:
  - Valid DNS name, all lower case
  - Start with alphanumeric, no consecutive dashes
- Messages URI:
  `http://acct-name.queue.core.windows.net/qname/messages`

109

Message Attributes

- MessageID
- PopReceipt
- VisibilityTimeout
  - Temporarily invisible after receive
  - Processing includes delete
- Messagettl
  - Default 7 days

110

REST API

- List queues

```
GET acct-name.queue.core.windows.net?
comp=list
[&prefix=prefix]
[&marker=marker]
[&maxresults=N]
```

111

REST API: Queues

Operation	HTTP Verb	URI
Create queue	PUT	http://*acct-name*.queue.core.windows.net/*qname*
Delete queue	DELETE	http://*acct-name*.queue.core.windows.net/*qname*
Get queue metadata	GET/ HEAD	http://*acct-name*.queue.core.windows.net/*qname*?comp=metadata
Set queue metadata	PUT	http://*acct-name*.queue.core.windows.net/*qname*?comp=metadata

112

REST API: Messages

Operation	HTTP Verb	URI
Put message	POST	http://*acct-name*.queue.core.windows.net/*qname*/messages
Get message	GET	http://*acct-name*.queue.core.windows.net/*qname*/messages
Peek messages	GET	http://*acct-name*.queue.core.windows.net/*qname*/messages?peekonly=true
Delete message	DELETE	http://*acct-name*.queue.core.windows.net/*qname*/*message-id*/popreceipt=*value*
Clear messages	DELETE	http://*acct-name*.queue.core.windows.net/*qname*/messages

113

Storage Client API

- Instantiate storage account from config file:

```
CloudStorageAccount account =
  CloudStorageAccount
    .FromConfigurationSetting
        (configurationSettingName);
```

- Create client:

```
CloudQueueClient client =
  account.CreateCloudQueueClient();
```

- List queues:

```
IEnumerable<CloudQueue> queues =
  client.ListQueues();
```

114

Storage Client API

- Create queue:

```
CloudQueue queue =
  client.GetQueueReference(queueName).Create();
```

- Add Message:

```
client
  .GetQueueReference(queueName)
    .AddMessage(queueMessage);
```

- Get Messages:

```
client
  .GetQueueReference(queueName)
    .GetMessages(numMessages,
                 TimeSpan.FromSeconds(visTimeout));
```

115

Asynchronous API

- Synchronous API call
  - Asychronous REST call to queue storage
  - Thread blocks waiting for response
  - Response waiting for thread to be become available
  - Deadlock!
- Workaround: Asynchronous API

116

Asynchronous API

```
CloudQueue q =
  client.GetQueueReference(queueName);

q.BeginAddMessage(queueMessage,
    TimeSpan.FromSeconds(ttlsecs),
    Completion,
    q);

public void Completion(IAsyncResult result) {
  var qc = result.AsyncState as CloudQueue;
  qc.EndAddMessage(result);
}
```

117

Asynchronous API

```
CloudQueue q =
  client.GetQueueReference(queueName);
using (System.Threading.ManualResetEvent evt =
       new System.Threading.ManualResetEvent(false))
{
  q.BeginAddMessage(queueMessage,
    TimeSpan.FromSeconds(ttlsecs),
    new AsyncCallback(result =>
        { var qc = result.AsyncState as CloudQueue;
          qc.EndAddMessage(result);
          evt.Set();
        }), q
    );
  evt.WaitOne();
}
```

118

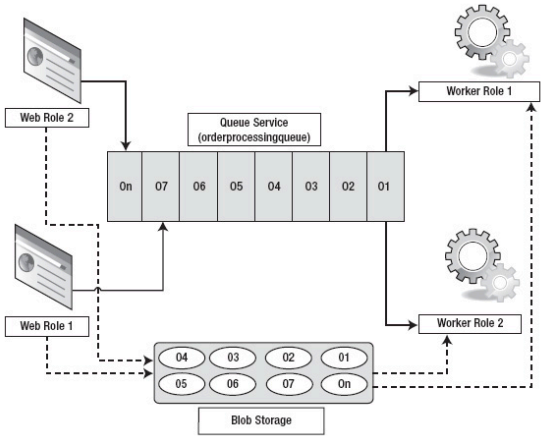
Asynchronous API

```
IEnumerable<CloudQueueMessage> ret = null;
using (System.Threading.ManualResetEvent evt =
       new System.Threading.ManualResetEvent(false))
{
  q.BeginGetMessages(numberOfMessages,
    new AsyncCallback(result =>
        { var qc = result.AsyncState as CloudQueue;
          ret = qc.EndGetMessages(result);
          evt.Set();
        }), q
    );
  evt.WaitOne();
}
return ret;
```

119

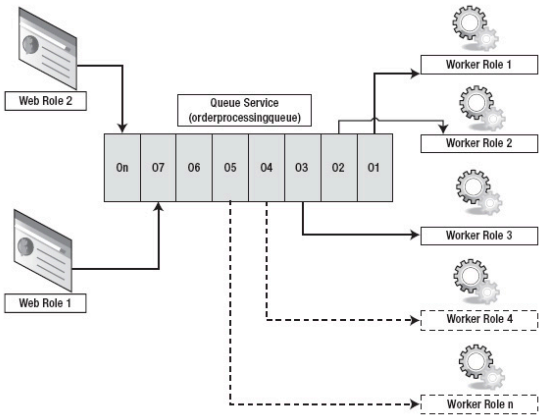
Queue Scenarios

- Scenario #1: Azure Web and Worker Roles
  - E.g. order processing



Queue Scenarios

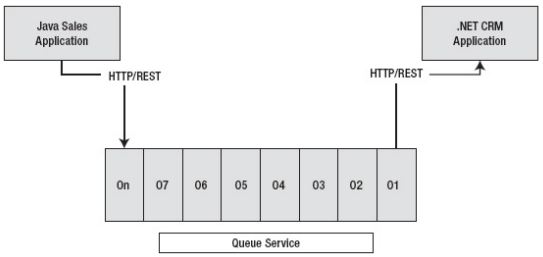
- Scenario #2: Worker Role Load Distribution
  - Queue as capacity indicator



121

Queue Scenarios

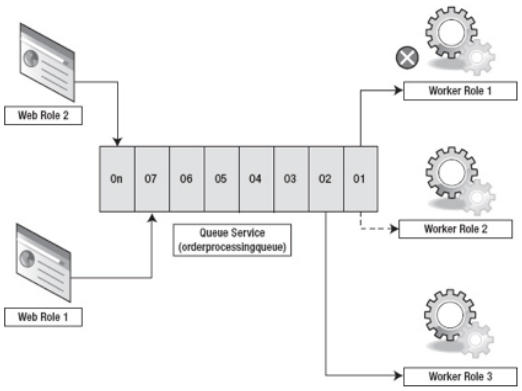
- Scenario #3: Interoperable Messaging
  - REST API



122

Queue Scenarios

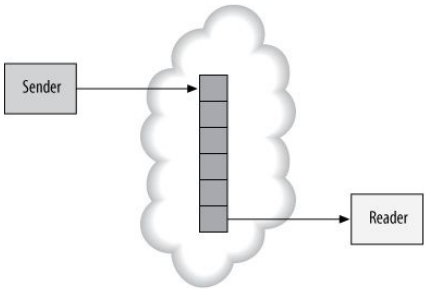
- Scenario #4: Guaranteed Processing
  - Set visibility timeout larger than average processing time
  - Don't delete until processed
  - Design Worker roles to be idempotent



SERVICE BUS BUFFERS

Service Bus Buffers

- Store and forward via service bus
- Decouple client from service
- Scenarios:
  - Buffer for shaky connection
  - One-way calls with response buffer



125

APPFABRIC MESSAGING: QUEUES AND TOPICS

126

Queues and Topics

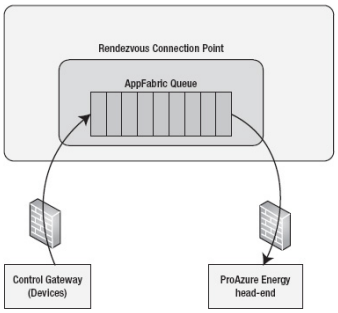
- AppFabric Message Buffer: temporary cache
- Azure Storage Queues: limited functionality

- Service Bus Queues
  - Persistent store for messages
- Service Bus Topics
  - Publish/subscribe

127

Service Bus Queues

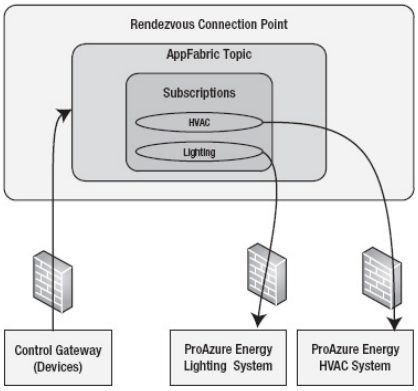
- Delivery patterns
  - At least once: `PeekLock`
  - At most once: `ReadAndDelete`
- .NET API and REST API
- Detection of duplicate
- Dead-letter queue
- Scheduled message delivery



128

Topics

- Organize head-end servers based on device topics
  - HVAC notifications
  - Lighting notifications



129

Subscription Rules

- SQLFilterExpression
  - Filter notifications based on SQL
- CorrelationFilterExpression
  - GUID-based correlation id
  - X_MS_CORRELATION_ID header in message

130

Creating a Queue

```
var baseAddress =
  RoleEnvironment.GetConfigurationSettingValue
    ("namespaceAddress");
Uri namespaceAddress =
  ServiceBusEnvironment.CreateServiceUri
    ("sb", baseAddress, string.Empty);

NameSpaceManager namespaceManager =
  new NamespaceManager(namespaceAddress,
        TokenProvider.CreateSharedSecretTokenProvider
            (ISSUER-NAME, ISSUER-KEY));

var queueDescription =
  namespaceManager.CreateQueue("energyqueue");
```

131

Subscribing to Topics

```
var topicDescription =
  nameSpaceManager.CreateTopic("energytopic");

var hvacSubscription =
  nameSpaceManager.CreateSubscriptionTopic
     (topicDescription.Path,
      "HVACSubscription",
      new SqlFilter("messageType='hvac'"));

var lightingSubscription =
   nameSpaceManager.CreateSubscriptionTopic
      (topicDescription.Path,
       "LightingSubscription",
       new SqlFilter("messageType='lighting'"));
```

132

Foundational Components

```
// Create MessagingFactory for this namespace
Uri serviceUri =
  ServiceBusEnvironment.CreateServiceUri("sb",
         ServiceNamespace, string.Empty);

TokenProvider credentials =
  TokenProvider.CreateSharedSecretTokenProvider
            (ISSUER-NAME, ISSUER-KEY));

MessagingFactory factory =
  MessagingFactory.Create(serviceUri,credentials);
```

133

Foundational Components

```
// Create Queue Client with at-least-once semantics
QueueClient queueClient =
  factory.CreateQueueClient("energyqueue",
                            ReceiveMode.PeekLock);

// Create Topic Client
TopicClient topicClient =
  factory.CreateTopicClient("energytopic");
```

134

Sending and Receiving: Queues

```
// Send message
BrokeredMessage message =
  new BrokeredMessage("Test message");
queueClient.Send(message);

// Receive message
QueueClient queueClient =
  factory.CreateQueueClient
                    (queueName,
                     ReceiveMode.PeekLock);
// check for a message, wait 5 seconds
// if queue is empty
BrokeredMessage receivedMessage =
  queueClient.Receive(new TimeSpan(0, 0, 5));
```

135

Sending and Receiving: Topics

```
// Publish topic
BrokeredMessage message =
  new BrokeredMessage("Test message");
Message.Properties["messageType"] = messageType;
topicClient.Send(message);

// Receive topic
SubscriptionClient hvacSubscriptionClient =
  factory.CreateSubscriptionClient
                    ("energytopic",
                     "HVACSubscription",
                     ReceiveMode.PeekLock);
BrokeredMessage message =
  hvacSubscriptionClient.Receive(new TimeSpan(0, 0, 5));
...
message.Complete();
```

136

Sending and Receiving: Topics

```
// Publish topic
BrokeredMessage message =
  new BrokeredMessage("Test message");
Message.Properties["messageType"] = messageType;
topicClient.Send(message);

// Receive topic
SubscriptionClient hvacSubscriptionClient =
  factory.CreateSubscriptionClient
                    ("energytopic",
                     "HVACSubscription",
                     ReceiveMode.ReceiveAndDelete);
BrokeredMessage message =
  hvacSubscriptionClient.Receive(new TimeSpan(0, 0, 5));
// No need to complete
```

137

DeadLetter Channel

- One subchannel for each queue or topic
  - Expired message
  - Max delivery count exceeded
  - Filter evaluation exception

138

DeadLetter Channel

- DeadLetter messages for a queue:

```
string dlqPath =
  queueClient.FormatDeadLetterPath("energyqueue");

QueueClient dlqClient =
  factory.CreateQueueClient(dlqPath,
            ReceiveMode.ReadAndDelete);

BrokeredMessage m;
while ((m = dlqClient.Receive()) != null) {
  Console.WriteLine("{0}",
            m.Properties("DeadLetterReason"));
}
```

139

DeadLetter Channel

- DeadLetter messages for a topic:

```
SubscriptionClient dlqSubClient =
  factory
    .CreateSubscriptionClient("energytopics",
            "HVACSubscription/$DeadLetterQueue");
            ReceiveMode.ReadAndDelete);

BrokeredMessage m;
while ((m = dlqSubClient.Receive()) != null) {
  Console.WriteLine("{0}",
            m.Properties("DeadLetterReason"));
}
```

140

Service Bus Queues vs Azure Storage Queues

- AppFabric extensions:
  - WCF binding (incl bidirectional TCP)
  - Poison message handling
  - Dead-lettering
  - Transactions
  - Groups
  - Sessions
  - Message deferral/sequencing
  - Authentication via ACS

141