Microsoft Azure

Azure Study Group
AZ-300 - Microsoft Azure Architect Technologies

Jeff Wagner

Partner Technology Strategist

Microsoft Azure

Develop for the Cloud (20-25%)

Agenda

1. Series Agenda
2. Speaker Introduction
3. Feedback Loop
4. Objective Review
5. Open Mic

Series Agenda

1	Deploy and Configure Infrastructure (25-30%)
2	Implement Workloads and Security (20-25%)
3	Create and Deploy Apps (5-10%)
4	Implement Authentication and Secure Data (5-10%)
5	Develop for the Cloud (20-25%)

https://aka.ms/azurecsg

Series Agenda

1	Deploy and Configure Infrastructure (25-30%)
2	Implement Workloads and Security (20-25%)
3	Create and Deploy Apps (5-10%)
4	Implement Authentication and Secure Data (5-10%)
5	Develop for the Cloud (20-25%)

https://aka.ms/azurecsg

Speaker Introduction - Jeff Wagner

- Partner Technology Strategist based in Atlanta
- 21+ years with Microsoft, more in the industry
- Been working with Microsoft Azure when we weren't sure if the was called Windows A*zure* or Windows *Azure*
- Constant learner - *Ancora Imparo*

Microsoft Azure

Feedback Loop

Objectives

Configure a message-based integration architecture

May include but not limited to: Configure an app or service to send emails, Event Grid, and the Azure Relay Service; create and configure Notification Hub, Event Hub, and Service Bus; configure queries across multiple products

Develop for autoscaling

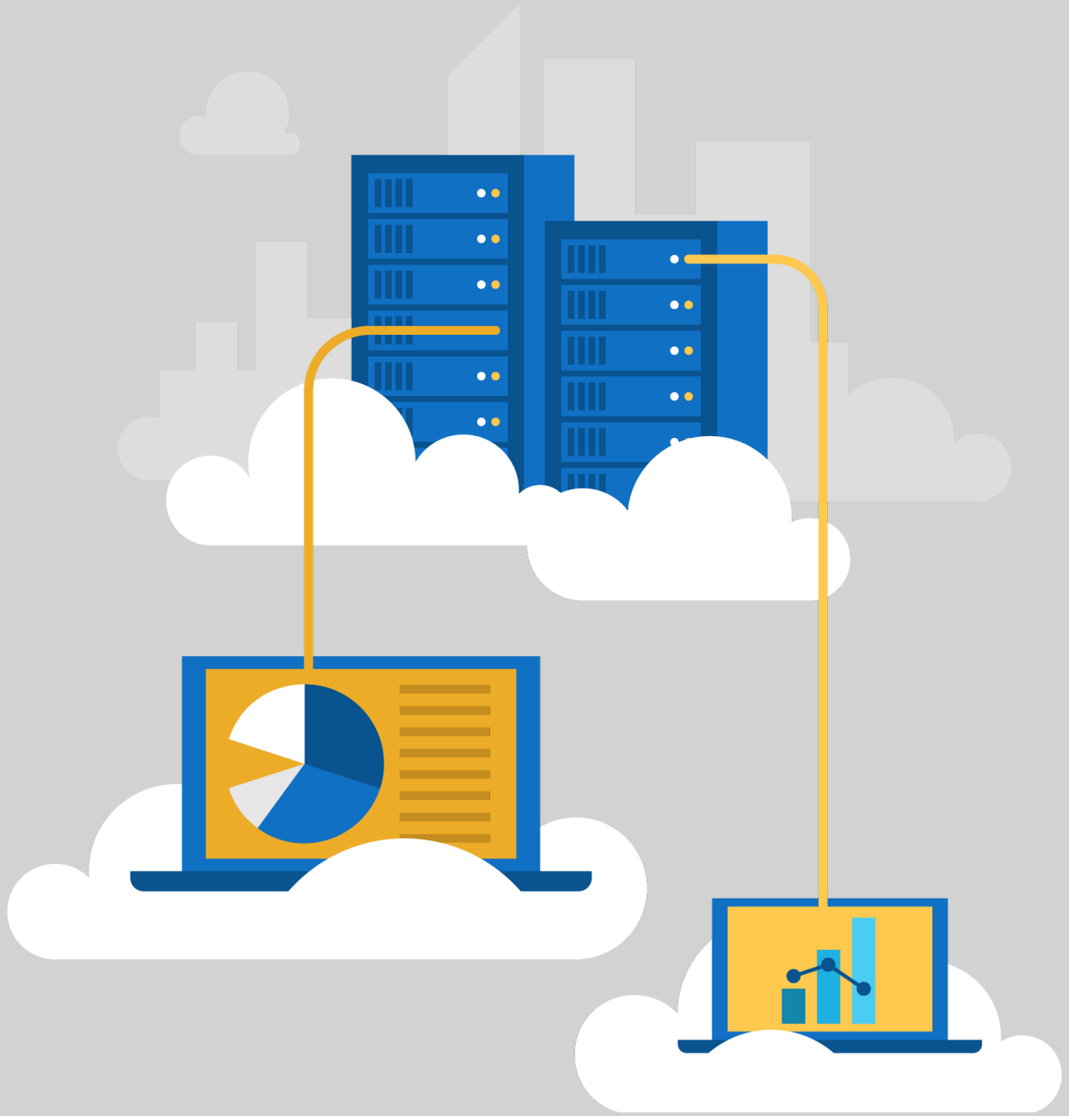
May include but not limited to: Implement autoscaling rules and patterns (schedule, operational/system metrics, code that addresses singleton application instances); implement code that addresses transient state

Microsoft

Configuring a Message-Based Integration Architecture

Microsoft

Configure an app or service to send emails



SendGrid

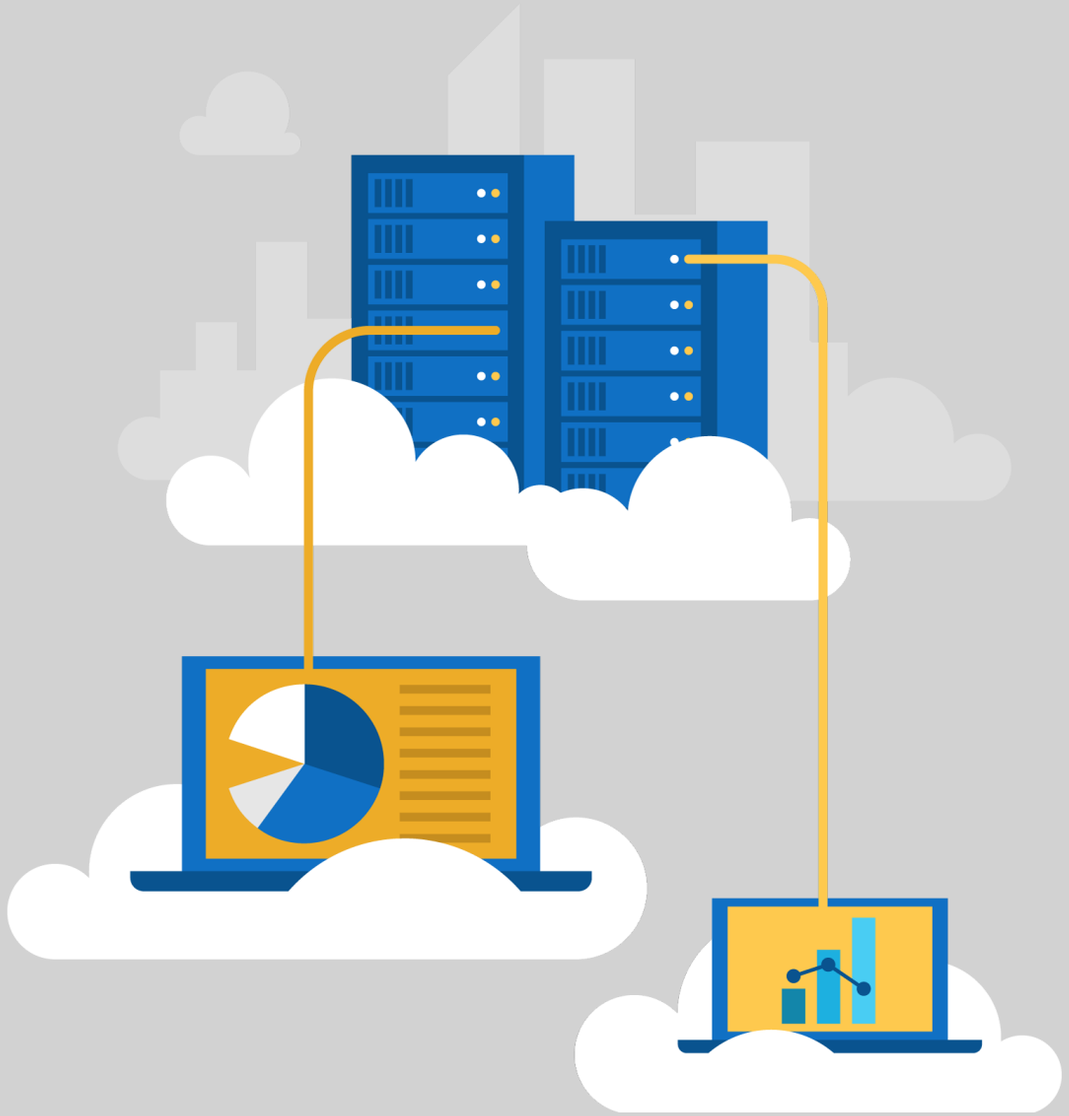
- A third party, cloud-based messaging service, which offers:
  - **deep level of integration with the Azure platform and the Azure portal**
  - **transactional email delivery**
  - **scalability based on email volume**
  - **real-time analytics**
  - **a flexible API to enable custom integration scenarios.**

Sending emails through SendGrid by using C#

- Based on an open source library:
  - **Hosted on GitHub at https://github.com/sendgrid/sendgrid-csharp**
  - **Available as a NuGet package**
- Implementation steps:
  - **Create a new instance of the SendGridClient class**
    - **Pass a string parameter containing API Key**
  - **Use class methods to perform common tasks:**
    - **SendGridClient.SendEmailAsync to send a simple email**
    - **AddAttachment to attach a file to an email message**

Microsoft

Configure an event publish and subscribe model



Event-driven architecture

- Consists of event producers and event consumers:
  - **Facilitates processing of events in near-real time**
  - **Decouples producers from consumers**
  - **Decouples consumers from each other**
  - **Allows all consumers to see all events**
- Offers several Azure-based implementations:
  - **Simple event processing: Each event triggers immediately an action in a consumer:**
    - **e.g. Azure Functions with an Azure Service Bus trigger**
  - **Complex event processing: Consumers process a series of events, looking for patterns:**
    - **e.g. Azure Stream Analytics or Apache Storm**
  - **Event stream processing: A stream processor processes or transform the stream:**
    - **e.g. Azure IoT Hub or Apache Kafka**

Azure Event Grid

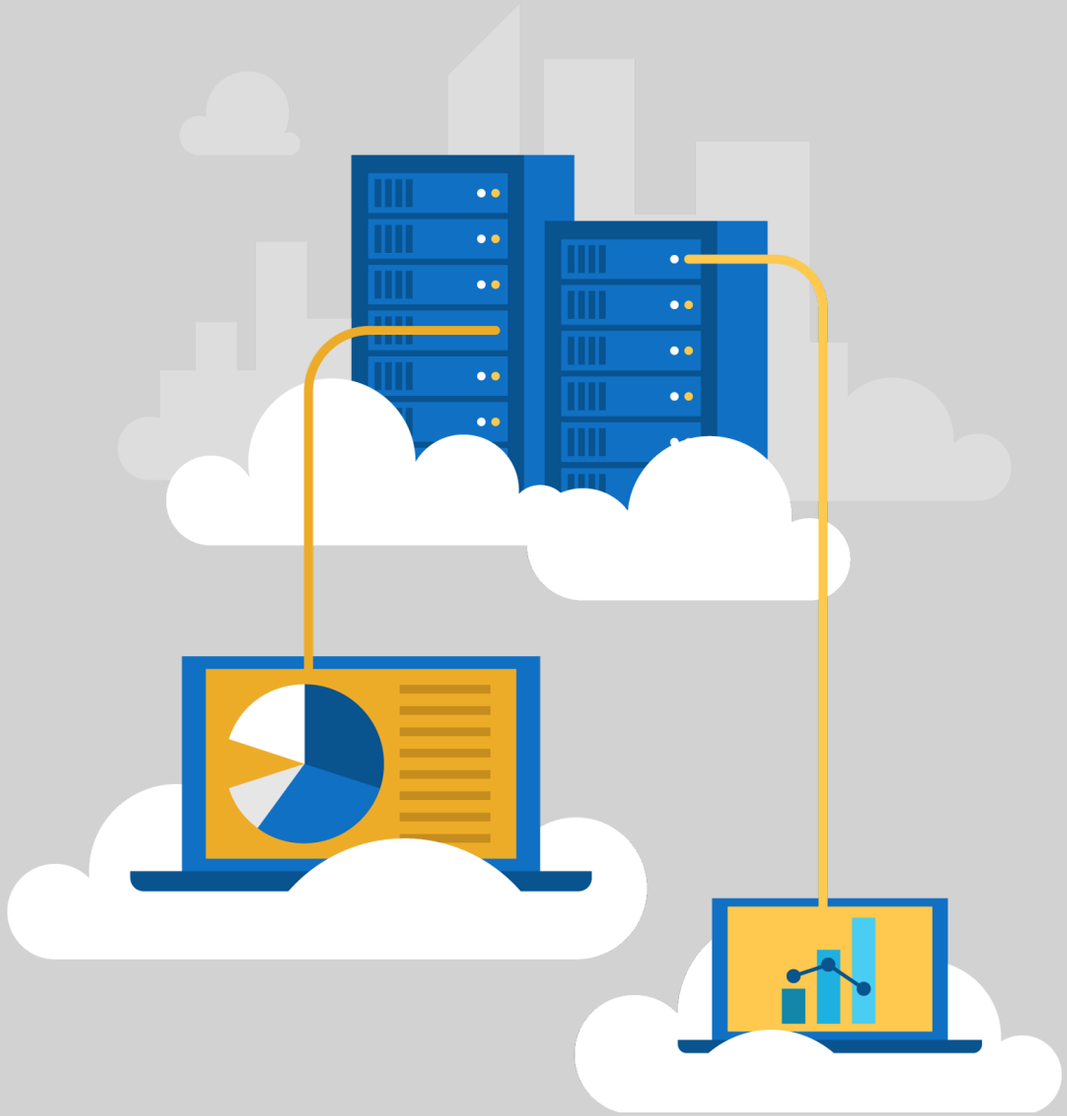
- Simplifies building applications with event-based architectures:
  - **Reduces implementation to:**
    - **Selecting an Azure resource to subscribe to**
    - **Providing the event handler or a webhook endpoint to send the event to**
  - **Offers built-in support for events originating from a range of Azure services, including:**
    - **Azure subscription management operations, custom topics, Azure Event Hubs, IoT Hub, Azure Media Services, Resource group management operations, Service Bus, Azure Blob storage, General Purpose v2 storage**
  - **Integrates with a range of Azure services that provide event handling:**
    - **Azure Automation, Azure Functions, Event Hubs, Hybrid Connections, Azure Logic Apps, Microsoft Flow, Azure Queue storage**

Subscribing to Blob storage events using Azure CLI

- Implementation steps:
  - **Create an Azure resource group:**
    - **az group create --name DemoGroup --location eastus**
  - **Create an Azure Storage account (Blob storage or General Purpose):**
    - **az storage account create --name demostor --location eastus --resource-group DemoGroup --sku \ Standard_LRS --kind BlobStorage --access-tier Hot**
  - **Identify the resource ID of the storage account:**
    - **storageid=$(az storage account show --name demostor --resource-group DemoGroup **
    - **--query id --output tsv)**
  - **Create a subscription:**
    - **az eventgrid event-subscription create --resource-id \$storageid –name **
    - **contosostoragesub --endpoint https://contoso.com/api/updates**

Microsoft

Configure the Azure Relay service



Azure Relay service

- Facilitates hybrid connectivity:
  - **Securely connects public cloud to services within a corporate enterprise network**
  - **Eliminates the need to open inbound firewall ports or change network infrastructure**
  - **Uses an outbound port to establish bidirectional socket dedicated to an internal service**
- Supports a variety of connectivity scenarios:
  - **Traditional one-way, request/response and peer-to-peer traffic**
  - **Event distribution to enable publish/subscribe scenarios**
  - **Bidirectional socket communication**
- Offers the Hybrid Connections capability, which additionally supports:
  - **WebSocket protocol**
  - **HTTP and HTTPs requests and responses**

Azure Relay service architecture

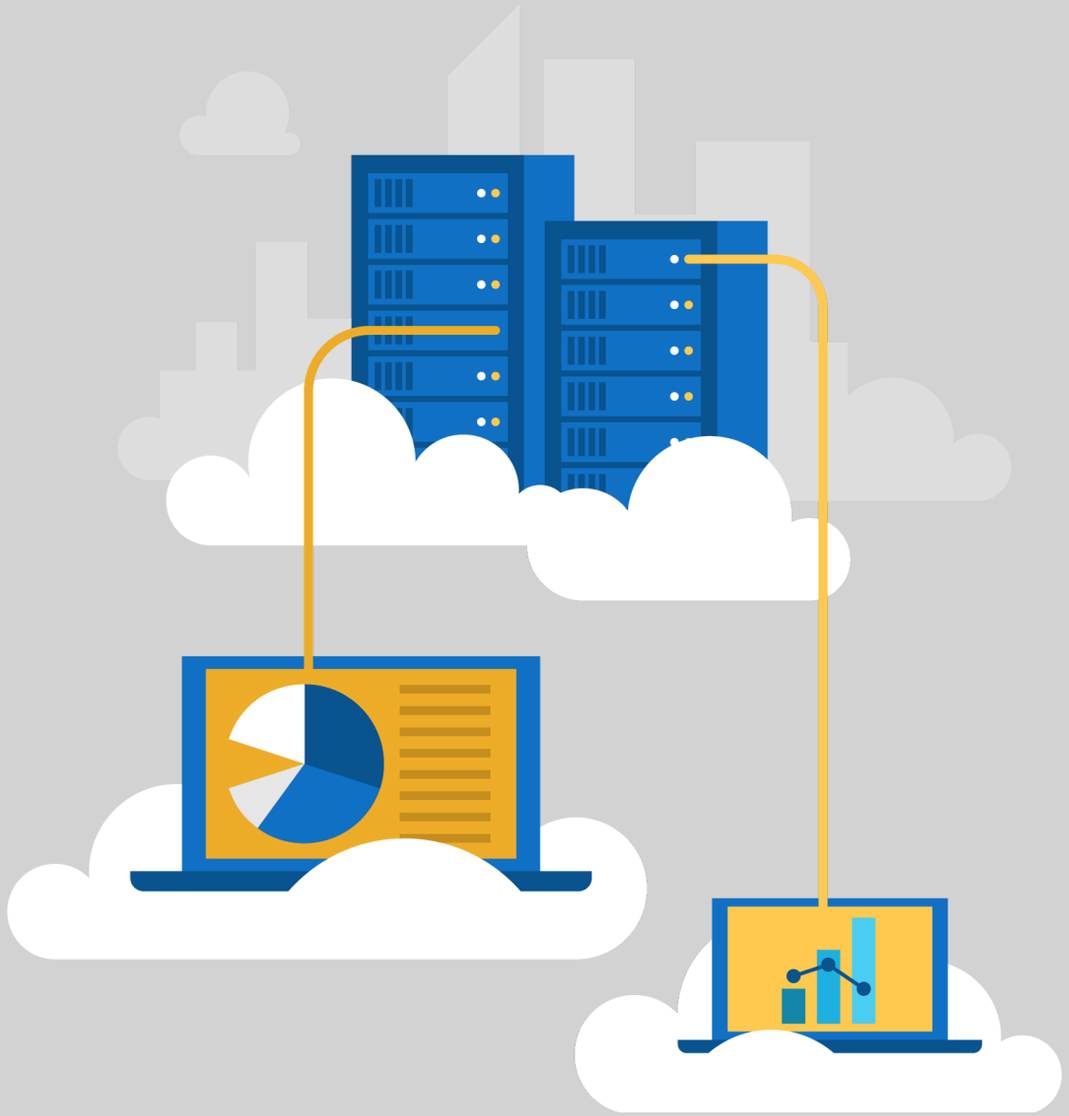
- **An Azure load balancer: routes requests to any of the gateway nodes.**
- **Gateway nodes: create new relays, forward connection requests to specific nodes that own a requested relays, and send rendezvous requests to listening clients.**
- **Listening clients: create temporary channels to gateway nodes in response to rendezvous requests and exchange messages via the gateway node.**
- **Communication flow:**
  - **A client (Client A) creates a listening request that is routed to a gateway.**
  - **The gateway creates a new relay.**
  - **A different client (Client B) creates a connection request.**
  - **The connection request from Client B is handled first by looking up the associated relay.**
  - **Once the correct relay has been identified, the request is forwarded to that specific relay.**
  - **A rendezvous request is sent to Client A.**
  - **Client A creates a temporary channel to Client B via the gateway that Client B used in its connection request.**
  - **Client B can receive messages from Client A directly from its original gateway.**
  - **Client A can receive messages from Client B via the original gateway and the established temporary channel.**

Using Azure Relay in Node.js

- Facilitated by:
  - **Node.js ws package WebSocket protocol client library**
  - **Node.js hyco-ws package for Hybrid Connections in Azure Relay, which extends ws:**
    - **Adds a new server class exported via require('hyco-ws').RelayedServer and a few helper methods**
    - **Provides hycows.RelayedServer class as an alternative to the ws.Server class. The RelayedServer constructor has two required arguments to establish a connection over the WebSocket protocol using Azure Relay:**
      - **server - The fully qualified URI for a Hybrid Connection name on which to listen.**
      - **token - Either a previously issued token string or a callback function that can be called to obtain such a token string.**
  - **Any libraries that support the WebSocket protocol in the language of your choice.**

Microsoft

Create and configure a notification hub



Azure Notification Hubs

- A scaled-out push engine that:
  - **Allows you to send notifications:**
    - **to practically any platform (iOS, Android, Windows, Kindle, BAIDU, etc.)**
    - **from practically any back end (cloud or on-premises).**
  - **Eliminates the complexity associated with implementing push notifications:**
    - **Offers multi-platform, scaled-out push notification infrastructure**
    - **Devices are responsible only for registering their PNS handles with a hub.**
  - **Facilitates a number of common notifications scenarios:**
    - **Sending breaking news notifications to millions with low latency**
    - **Sending location-based coupons to interested user segments**
    - **Sending event-related notifications for media/sports/finance/gaming applications**
    - **Pushing promotional content to applications to engage and market to customers**
    - **Notifying users of enterprise events, like new messages and work items**
    - **Sending codes for multi-factor authentication**

Configuring Notification Hubs in iOS

- Ensure that an Xcode application satisfies the following prerequisites:
  - **The Push Notifications capability must be enabled.**
  - **The files distributed in the Azure messaging framework must be included in the project.**
- Implement the following steps:
  - **add constants to HubInfo.h that will contain connection details for your notification hub**
  - **import the WindowsAzureMessaging/WindowsAzureMessaging and UserNotifications/UserNotifications directives into the project**
  - **add code to connect to the notification hub using the information stored in HubInfo.h**

Configuring Notification Hubs in Xamarin.Android

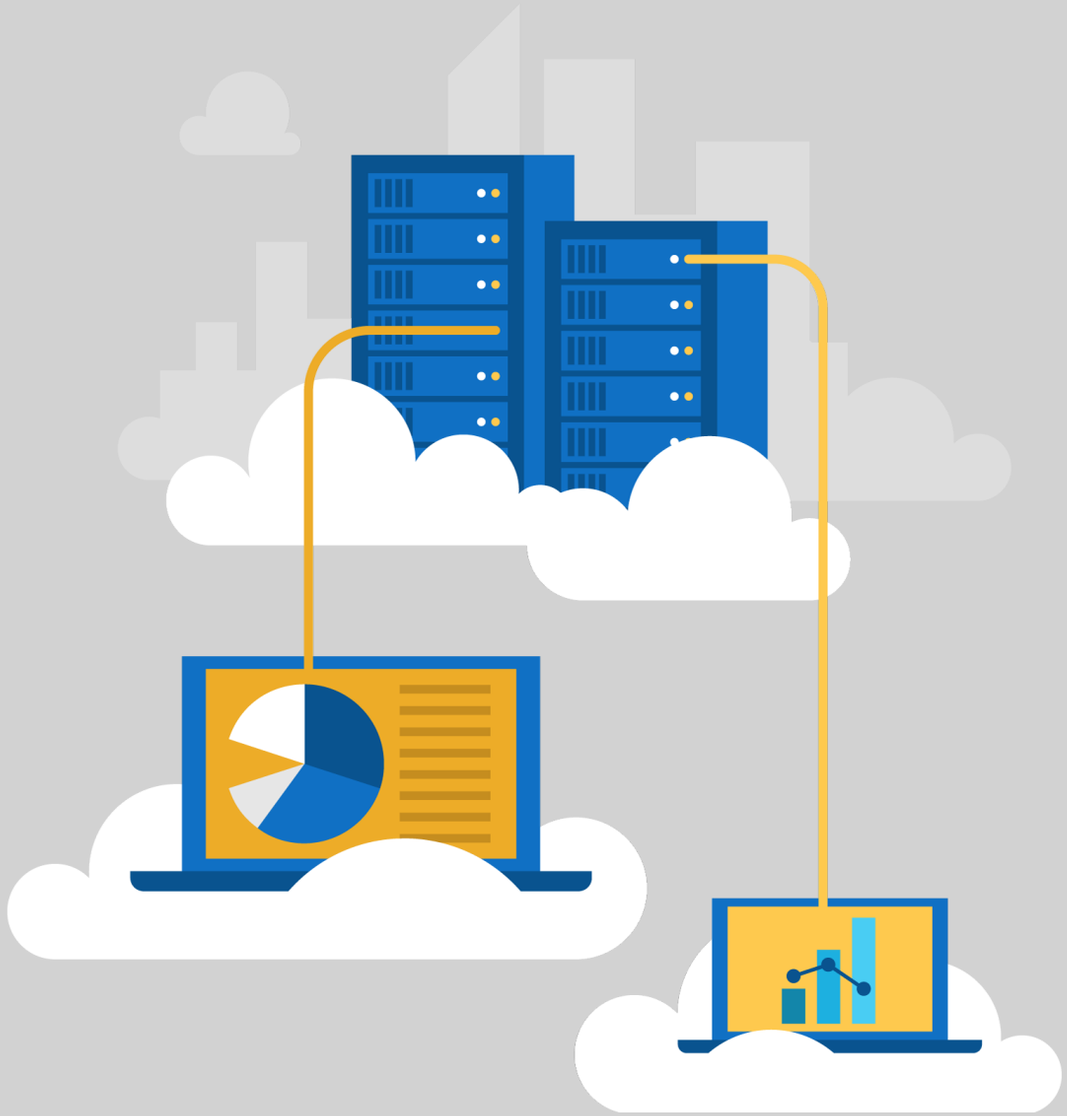
- Ensure that a Xamarin project satisfies the following prerequisites:
  - **The Xamarin.GooglePlayServices.Base, Xamarin.Firebase.Messaging, and Xamarin.Azure.NotificationHubs.Android NuGet packages must be installed.**
  - **The google-services.json file must be downloaded from the Google Firebase Console and then copied to the root of your project folder.**
  - **com.google.firebase.iid.FirebaseInstanceIdReceiver receiver must be registered to enable PNS registration and message receipt.**
- Implement the following steps:
  - **Add a C# class that will contain connection details for your notification hub**
  - **Create a C# class to manage PNS registration**
  - **Create a separate C# class to handle the receipt of a new message and display that message in the application's UI**

Sending messages from an application back end to Notification Hubs using C#

- Available via a number of client libraries:
  - **In .NET applications, the Microsoft.Azure.NotificationHubs NuGet package includes the NotificationHubClient class that can be used to send messages**
- Simplified by Azure Notification Hubs templates:
  - **Allow you to specify how a device should receive notifications**
  - **Facilitate sending notifications to multiple platforms at the same time without having to specify a native payload**

Microsoft

Create and configure an event hub



Azure Event Hubs

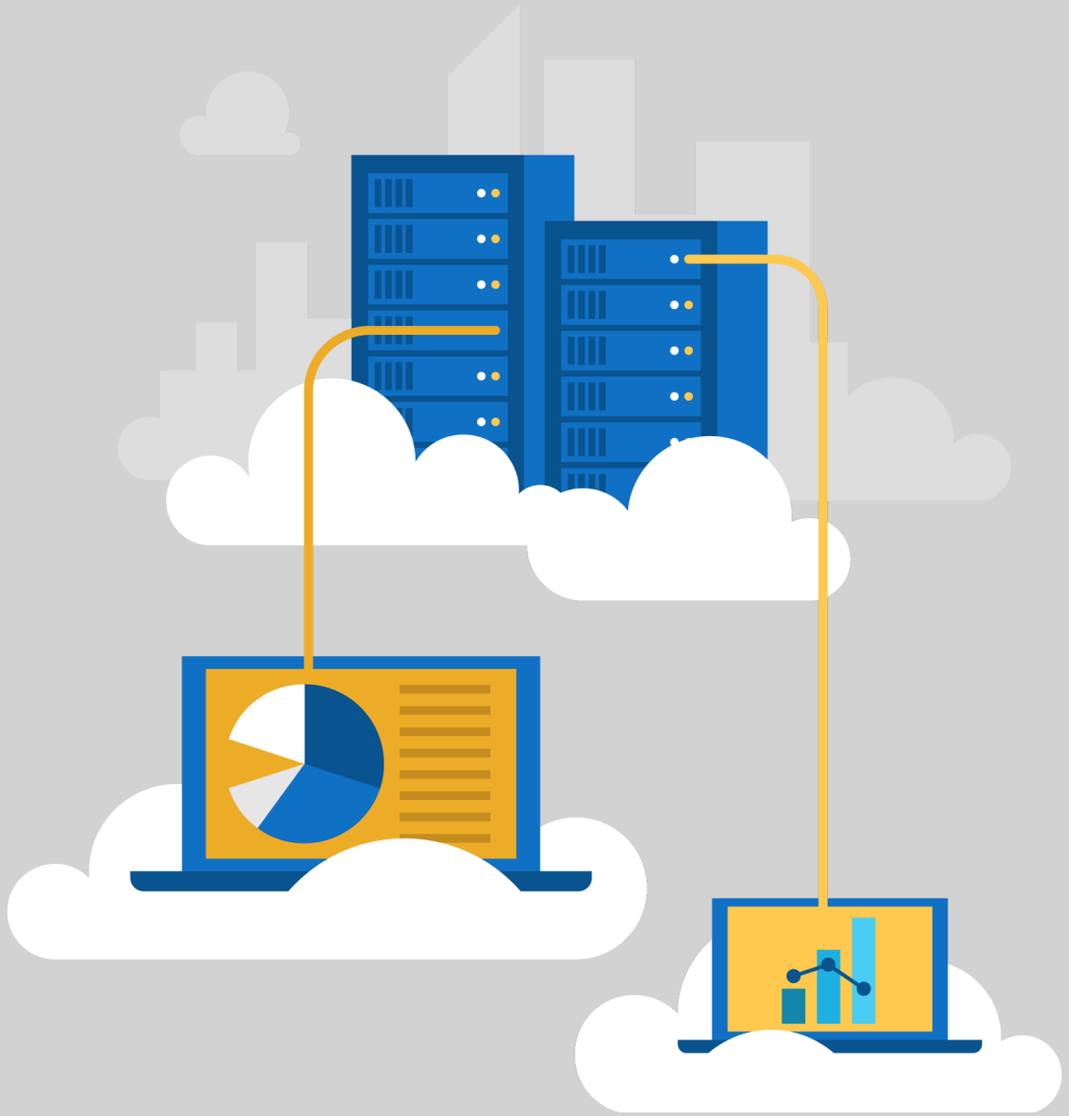
- A big data streaming platform and event ingestion service:
  - **Capable of receiving and processing millions of events per second**
  - **Facilitating processing and storing events, data, or telemetry produced by distributed software and devices**
  - **Providing low latency and seamless integration with data and analytics services inside and outside of Azure**
  - **Serving as an event ingestor in a pipeline between event producers and consumers**
  - **Consisting of the following main components:**
    - **Event producers**
    - **Partitions**
    - **Consumer groups**
    - **Throughput units**
    - **Event receivers**

Azure IoT Hub

- A managed service operating as a central message hub:
  - **Provides bidirectional communication between IoT application and IoT devices**
  - **Scales to millions of IoT devices of practically any type**
  - **Supports multiple messaging patterns, including:**
    - **device-to-cloud telemetry**
    - **file upload from devices**
    - **request-reply methods**
- Differs from Azure Event hubs in a number of aspects:
  - **IoT Hub:**
    - **developed specifically to address the unique requirements of connecting IoT devices**
    - **includes features that enrich the relationship between IoT devices and back-end systems**
  - **Event Hubs:**
    - **designed for big data streaming**
    - **offer a partitioned consumer model to scale out stream and integrate with big data and analytics services**

Microsoft

Create and configure a service bus



Azure Service Bus

- A messaging service that offers three communication mechanisms:
  - **Queues,** which allow one-directional communication. Each queue acts as an intermediary that stores sent messages until they are received. Each message is received by a single recipient.
  - **Topics,** which provide one-directional communication using subscriptions. A single topic can have multiple subscriptions. Like a queue, a topic acts as a broker, but each subscription can use a filter to receive only messages that match specific criteria.
  - **Relays,** which provide bidirectional communication. Unlike a queue or a topic, a relay doesn't store in-flight messages, it just passes them on to the destination application.
- Namespace is the scoping container for a Service Bus implementation:
  - **Multiple queues, topics, and relays can reside within a single namespace.**
  - **Namespaces often serve as application containers.**
  - **Each namespace name must be globally unique.**

Service Bus queues

- Support a brokered, asynchronous messaging communication model:
  - **A message producer (sender) pushes a message to a queue and continues its processing.**
  - **A message consumer (receiver) pulls the message from the queue and processes it.**
- Can be used for a wide variety of scenarios:
  - **Communication between web and worker roles in a multi-tier Azure application.**
  - **Communication between on-premises apps and Azure-hosted apps in a hybrid solution.**
  - **Communication between the components of a distributed application running on-premises in different organizations or departments of an organization.**

Manipulating a Service Bus queue using Ruby

- Implementation steps:
  - **Download the Azure Ruby package:**
    - **gem install azure**
  - **Add the azure directive at the top of Ruby file where you want to reference the Azure Ruby libraries:**
    - **require "azure“**
  - **Create a connection to Service Bus using the client object**
  - **Send messages to a Service Bus queue by calling the send_queue_message() method on the Azure::ServiceBusService object:**
    - **Messages are Azure::ServiceBus::BrokeredMessage objects and have:**
      - **a set of standard properties (such as label and time_to_live)**
      - **a dictionary that is used to hold custom**
      - **application-specific properties**
      - **a body of arbitrary application data.**

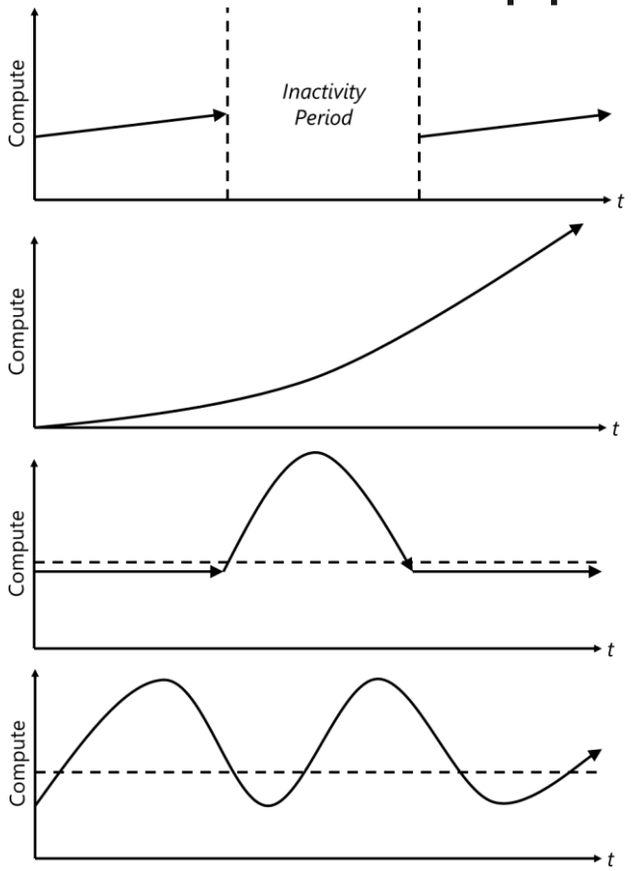
Microsoft

Developing for Autoscaling

Subtitle or speaker name

Computing patterns

- The most common computing patterns for cloud-based web apps:
  - **On and off:**
    - **Common for batch processing**
    - **Waste of overprovisioned capacity**
  - **Growing fast:**
    - **Represents a successful service**
    - **Presents growth challenges**
  - **Unpredictable bursting:**
    - **Unexpected peaks of demand**
    - **Waste of overprovisioned capacity**
  - **Predictable bursting:**
    - **Requires resource provisioning and deprovisioning**
    - **Introduced increased management complexity**



Scale and auto-scale

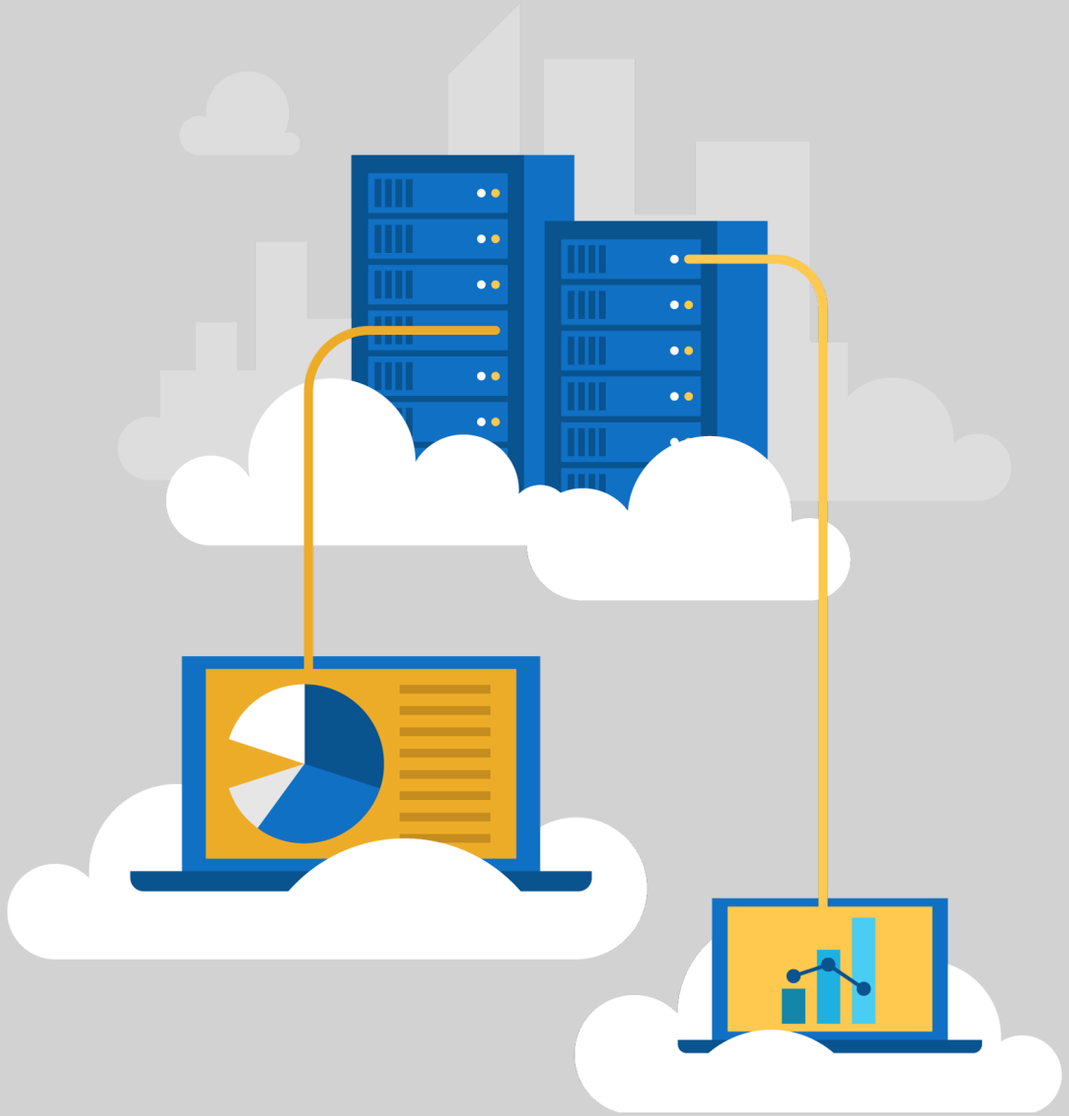
- Cloud offers elastic scaling by using its practically unlimited resources
- Auto-scaling is the ability to monitor workload usage and automatically scale according to changes in usage levels
- Many Azure services support manual and automatic scaling:
  - **infrastructure services such as Azure Virtual Machine Scale Sets**
  - **application services such as Azure App Service**
  - **database services such as Azure Cosmos DB.**

Auto-scale metrics

- App Service support autoscale:
  - **Requires service plans using Basic, Standard, or Premium pricing tiers**
  - **Increases or decreases the instance count within the same service plan**
  - **Supports a number of auto-scale metrics:**
    - **CPU: CpuPercentage**
    - **Memory: MemoryPercentage**
    - **Data in: BytesReceived**
    - **Data out: BytesSent**
    - **HTTP queue: HttpQueueLength**
    - **Disk queue: DiskQueueLength**
  - **Allow you to trigger alerts when the value of a metric crosses a custom threshold:**
    - **Send email notifications to the service administrator and co-administrators**
    - **Send email to additional email addresses that you specify**
    - **Call a webhook**
    - **Start the execution of an Azure runbook**

Microsoft

Implement code that addresses singleton application instances



Querying resources using Azure CLI

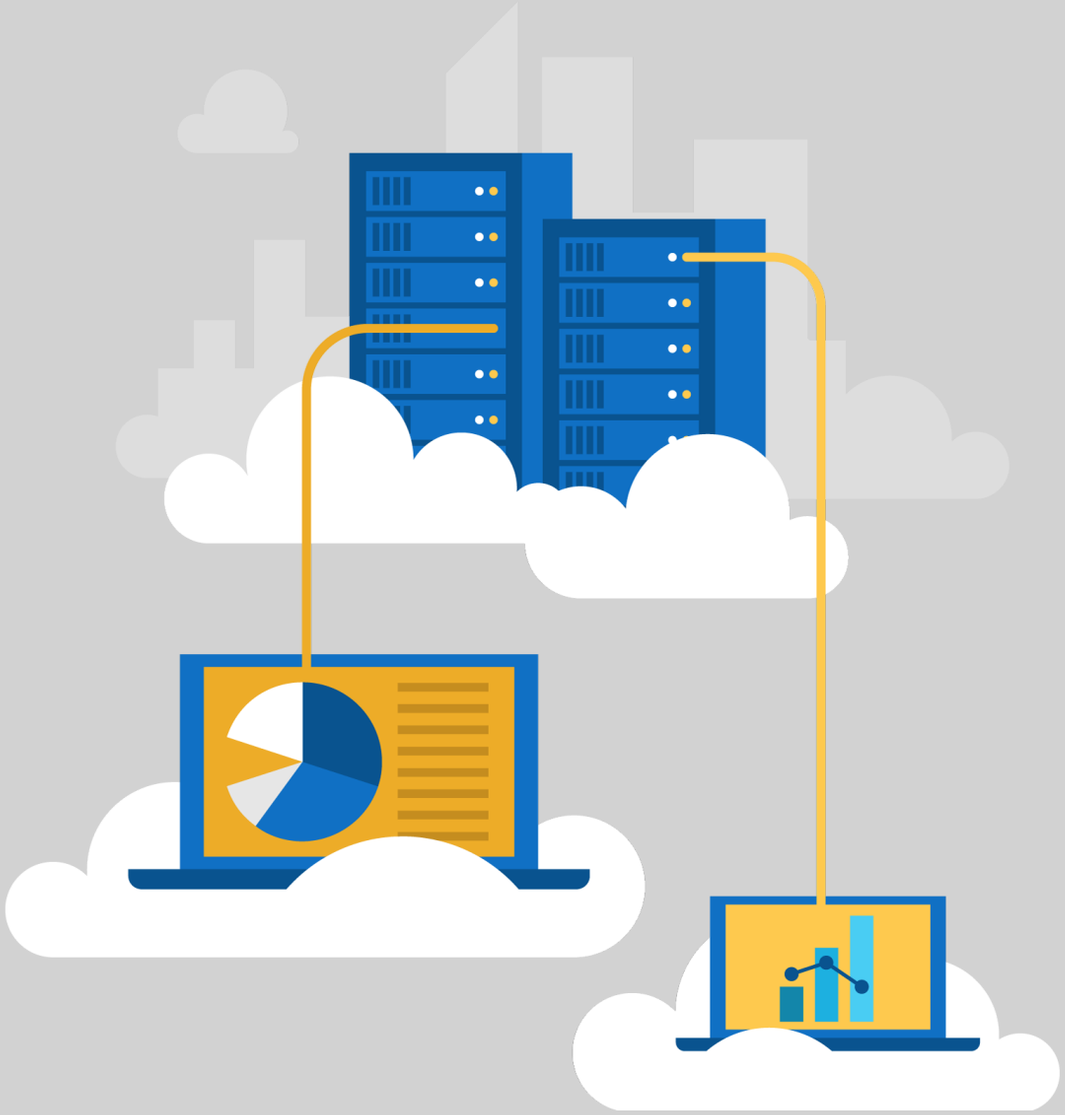
- Azure CLI is optimized for managing Azure resources:
  - **Supports the --query argument to execute a JMESPath query on command results:**
    - **A single result in the format of a JSON document**
    - **Multiple results in the format of a JSON array (you can flatten them by using the [] JMESPath operator)**
  - **For example, to list Azure VMs in the current subscription, run:**
    - **az vm list**
  - **To limit results to the image name and offer, run:**
    - **az vm list --query '[].{name:name, image:storageProfile.imageReference.offer}'**
  - **To list Azure VMs with the Windows Server OS installed, run:**
    - **az vm list --query "[?starts_with(storageProfile.imageReference.offer, 'WindowsServer')]“**
  - **To list name and vmID or Azure VMs with Ubuntu OS installed, run:**
    - **az vm list --query "[?starts_with(storageProfile.imageReference.offer, 'Ubuntu')].{name:name, id:vmId}"**

Querying resources using the fluent Azure SDK

- Implementation steps:
  - **Create an authenticated client by invoking the Azure.Authenticate static method:**
    - **The Authenticate method requires an authorization file necessary to access a target subscription:**
      - **Azure azure = Azure.Authenticate("azure.auth").WithDefaultSubscription();**
    - **You can generate the file by using the Azure CLI: az ad sp create-for-rbac --sdk-auth > azure.auth**
  - **Query resources by using properties of the IAzure interface of the Azure variable:**
    - **To list Azure VMs synchronously, use the VirtualMachines property:**
      - **azure.VirtualMachines**
    - **To list Azure VMs synchronously, use the VirtualMachines property:**
      - **var vms = await azure.VirtualMachines.ListAsync();**
        foreach(var vm in vms)
        {
        Console.WriteLine(vm.Name);
        }
    - **To determine an IP address of an Azure VM, use the IVirtualMachine and related interfaces:**
      - **INetworkInterface targetnic = targetvm.GetPrimaryNetworkInterface();**
      - **INicIPConfiguration targetipconfig = targetnic.PrimaryIPConfiguration;**
      - **IPublicIPAddress targetipaddress = targetipconfig.GetPublicIPAddress();**

Microsoft

Implement code that addresses a transient state

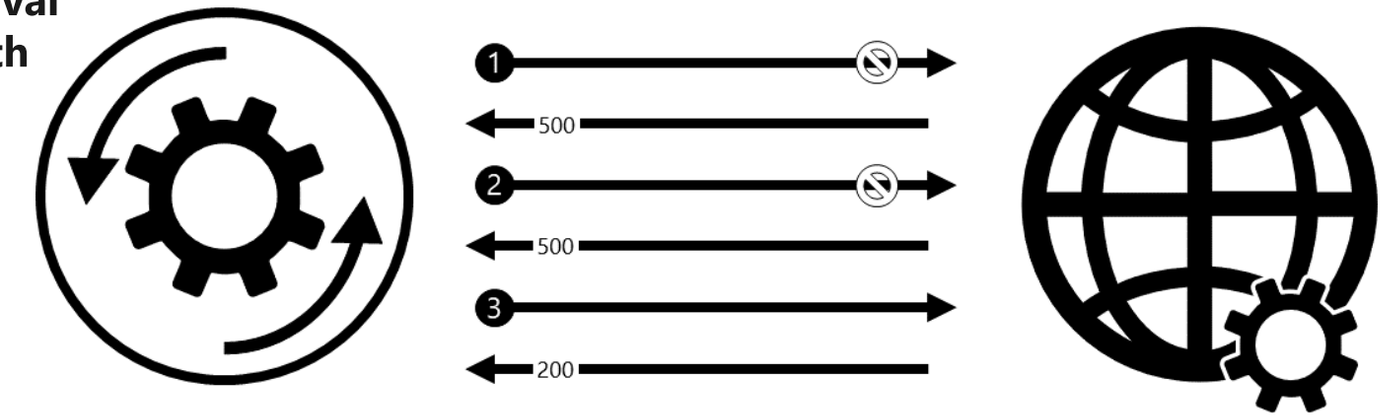


Transient errors

- Transient faults are typically self-correcting and include:
  - **momentary loss of network connectivity to components and services**
  - **temporary unavailability of a service**
  - **timeouts that occur when a service is busy**
- Applications that use cloud services should be able to handle gracefully transient faults:
  - **If the action that triggered a fault is repeated after a suitable delay, it is likely to succeed**

Handling transient errors

- Applications can handle a failure by using the following strategies:
  - **Cancel: if the fault indicates that the failure is not transient**
  - **Retry: If the specific fault reported is unusual or rare**
  - **Retry after delay: If the fault is caused by a common connectivity or busy failure:**
    - **the period between retries should be chosen to result in an even distribution of requests from multiple instances of the application:**
    - **1.The application invokes an operation on a hosted service. The request fails, and the service host responds with HTTP response code 500 (internal server error).**
    - **2.The application waits for a short interval and tries again. The request fails with HTTP response code 500.**
    - **3.The application waits for a longer interval and tries again. The request succeeds with**
    - **HTTP response code 200 (OK).**



Handling transient errors in code

- **The OperationWithBasicRetryAsync method invokes an external service asynchronously through the TransientOperationAsync method.**
- **The details of the TransientOperationAsync method are service-specific**
- **The method is contained in a try/catch block wrapped in a for loop:**
  - **If the TransientOperationAsync method succeeds, the for loop exits.**
  - **If the TransientOperationAsync method fails, the catch block examines the reason for the failure. If it's believed to be a transient error, the code waits for a short delay before retrying the operation.**
  - **The for loop also tracks the number of times that the operation has been attempted.**
  - **If the code fails three times, the exception is assumed to be more long lasting.**
  - **If the exception is long lasting, the catch handler throws an exception.**

Detecting if an error is transient in code

- The IsTransient method checks for a specific set of exceptions that are relevant to the environment the code is run in:
  - **Specifics are context-dependent**

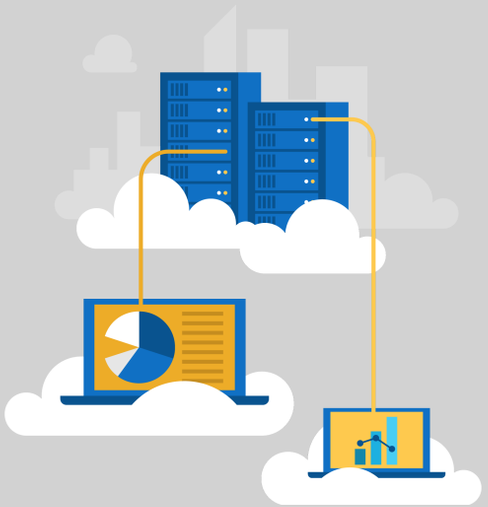
```
private bool IsTransient(Exception ex)
{
        if (ex is OperationTransientException)
                return true;

        var webException = ex as WebException;
        if (webException != null)
        {
                return new[] {
                        WebExceptionStatus.ConnectionClosed,
                        WebExceptionStatus.Timeout,
                        WebExceptionStatus.RequestCanceled
                }.Contains(webException.Status);
        }

        return false;
}
```

Questions?

Homework Assignment

GET EXAM READY!

Note the changes coming 2/22/19



Microsoft Azure

Open Mic