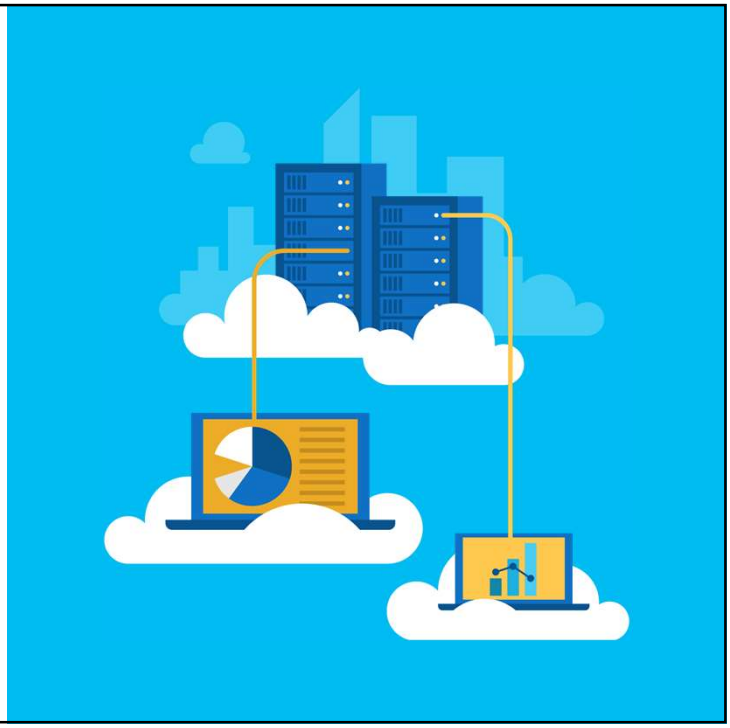
Microsoft

AZ-300T06 Module 01: Developing for Autoscaling

Subtitle or speaker name



1

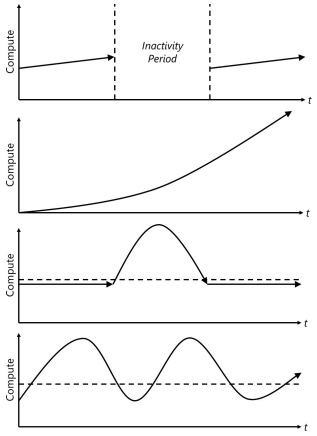
Module 01: Developing for Autoscaling

Lesson 01: Implement autoscaling rules and patterns

2

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**



3

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.**

4

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**

5

Module 01: Developing for Autoscaling

Lesson 02: Implement code that addresses a transient state

6

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**

7

Handling transient errors

Applications can handle a failure by using the following strategies:

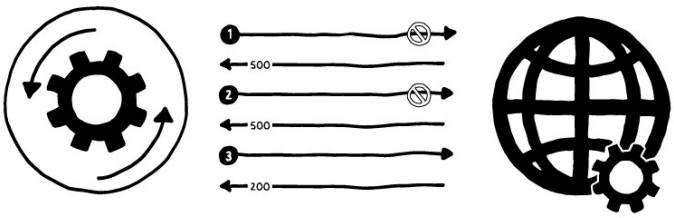
- **Cancel: if the fault indicates that the failure is not transient**
- **Retry: If the specific fault reported is unusual or rare**
- **Retry after delay: If the fault is caused by a common connectivity or busy failure:**
  - **The period between retries should be chosen to result in an even distribution 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).**



8

9