Module 2

Designing Cloud Applications for Resiliency

Module Overview

- Application Design Practices for Highly Available Applications
- Application Analytics
- Building High Performance Applications by Using ASP.NET
- Common Cloud Application Patterns
- Caching Application Data

Lesson 1: Application Design Practices for Highly Available Applications

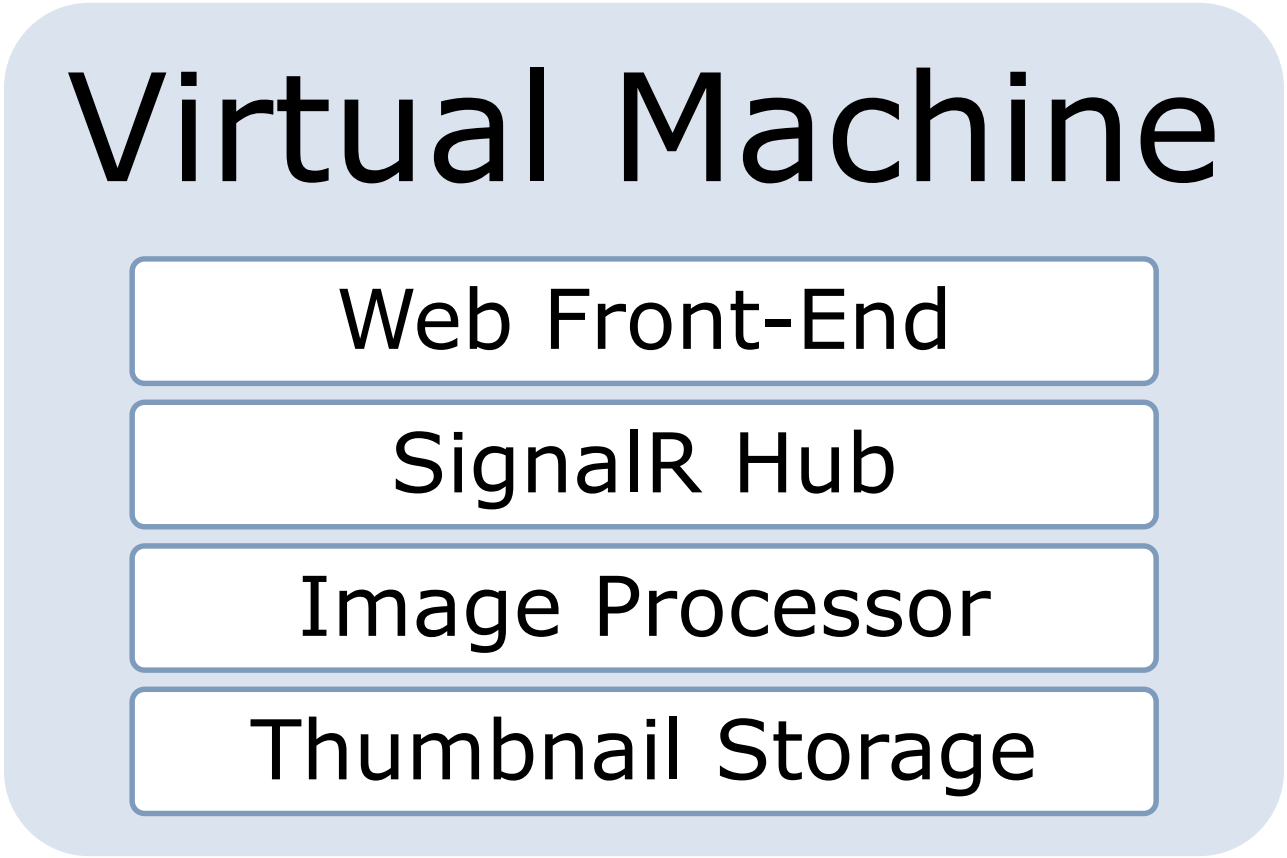
- Partitioning Workloads
- Load Balancing
- Transient Fault Handling
- Queues

Partitioning Workloads

- When designing web applications, split your business processes into Partitioning Workloads
- Partitioning Workloads:
  - Can be handled in modular websites, cloud services, or virtual machines
  - Provides the ability to scale the different components of your application in isolation

Partitioning Workloads (cont.)

- Example Photo Sharing Application

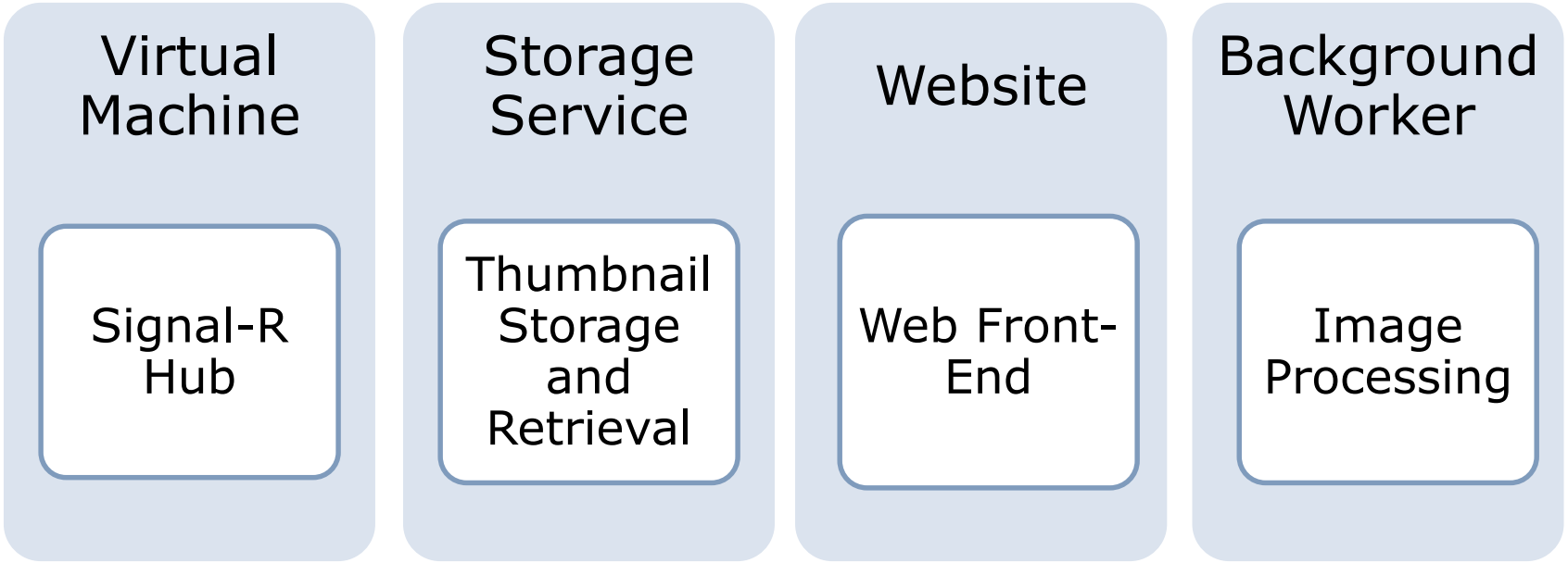


Partitioning Workloads (cont.)

- Example Photo Sharing Application
  - The Web Front-End shows thumbnails of images users have uploaded today
  - The application code takes an image that a user uploads, processes it into a thumbnail and then stores the thumbnail.
  - The SignalR hub notifies the client web browser that the image is finished processing

Partitioning Workloads (cont.)

- Example Photo Sharing Application



- Each component of the application can be scaled in isolation to meet its specific demand

Load Balancing

- Provide the same service from multiple instances and use a load balancer to distribute requests across all of the instances
- Considerations for selecting a load balancing strategy:
  - Hardware or software load balancers
  - Load balancing algorithms (round robin)
  - Load balancer stickiness

Load Balancing and Geographic Resiliency

- Load balancing can be combined with geographic redundancy to help achieve high availability
  - You can use a load balancer to direct client requests to their closest data center
    - Traffic Manager is often used for this task
  - A load balancer can be used to implement a failover scenario
    - When a data center or compute instance is down, clients can be directed to the next desirable instance that is available
- Designing a load balancing strategy along with distributing your application across data centers is key to high availability across the globe

Transient Fault Handling

- Transient faults can occur because of a temporary condition
  - Service is unavailable
  - Network connectivity issue
  - Service is under heavy load
- Retrying your request can resolve temporary issues that normally would crash an application
- You can retry using different strategies
  - Retry after a fixed time period
  - Exponentially wait longer to retry
  - Retry in timed increments

Transient Fault Handling (cont.)

- The Transient Fault Handling application block is a part of the Enterprise Library
- The Enterprise library contains a lot of code that is necessary to implement the pattern and the retry strategies
- Retry strategies are prebuilt for common scenarios including accessing Azure services
- You can build custom strategies and extend the library for the unique needs of your application

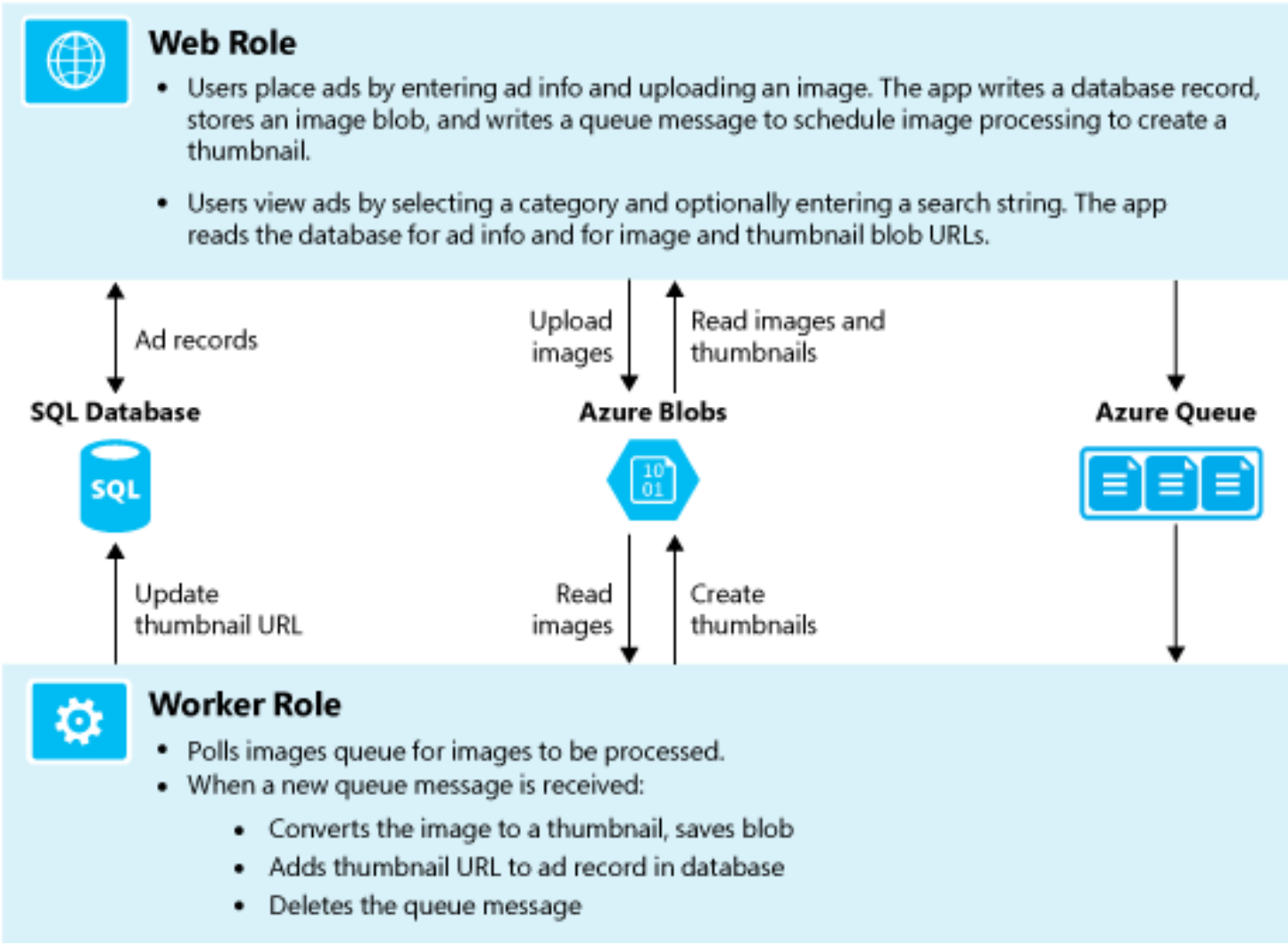
Queues

- A modular web application can behave like a monolithic application if each component relies on a direct two-way communication with a persistent connection
- Persistent queue messages allow your application to handle requests if one of your application components fail or is temporary unavailable
- An external queue allows your application to audit requests or measure your load without adding any overhead to the code of your primary application

Queues (cont.)

- Queues can be used to communicate in a disconnected manner between the different components of your application
  - If an instance of your application module fails, another instance can process the queue messages
  - Messages that fail to be processed can either be reinserted into the queue or not marked as complete by the client module
  - Messages can be measured or audited in isolation from your application
- Queues become very important when dealing with background processing (Worker roles and WebJobs)

Queues (cont.)


Web Role
Users place ads by entering ad info and uploading an image. The app writes a database record, stores an image blob, and writes a queue message to schedule image processing to create a thumbnail.
Users view ads by selecting a category and optionally entering a search string. The app reads the database for ad info and for image and thumbnail blob URLs.
Ad records
Upload images
Read images and thumbnails
SQL Database
Azure Blobs
Azure Queue
SQL
Update thumbnail URL
Read images
Create thumbnails
Worker Role
Polls images queue for images to be processed.
When a new queue message is received:
Converts the image to a thumbnail, saves blob
Adds thumbnail URL to ad record in database
Deletes the queue message

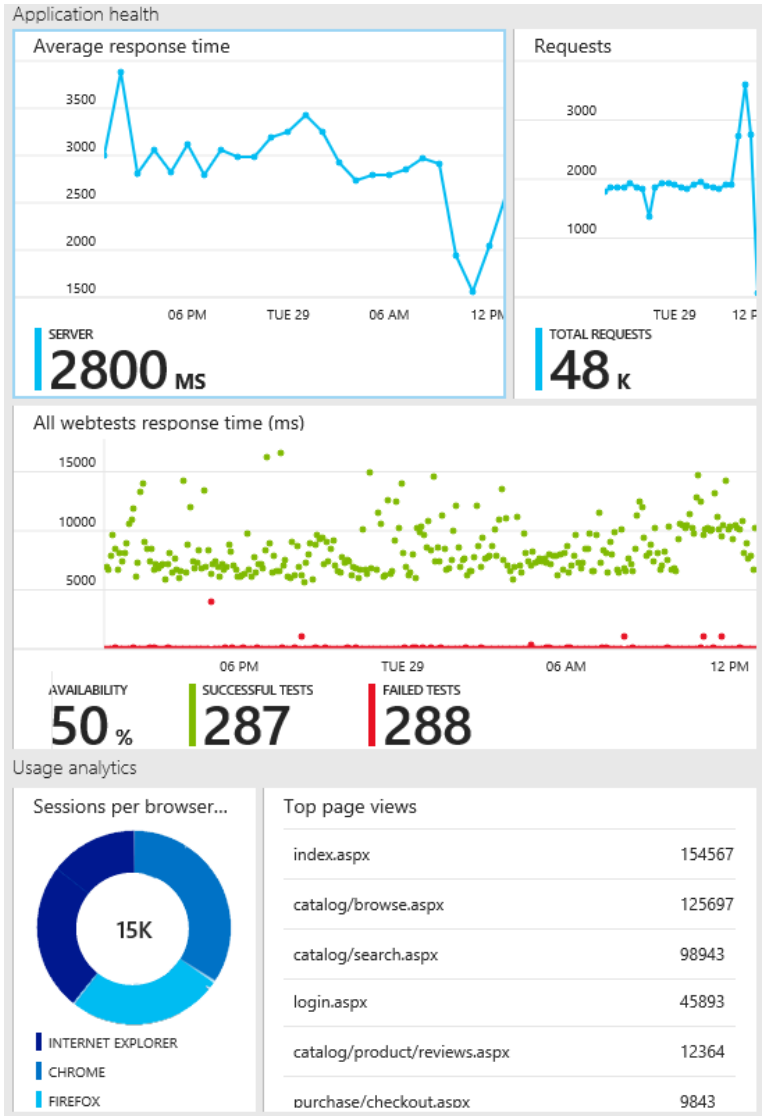
Lesson 2: Application Analytics

- Application Insights
- Demonstration: Monitoring a Web Application

Application Insights

- Application Insights is a analytics and monitoring service available for your applications
  - View exception stack traces
  - Monitor CPU and resource usage
  - Periodically test URLs from worldwide data centers
  - Monitor usage of your application and most popular requests
- It can be used with .NET or Java
- Applications do not specifically need to be hosted in Azure

Application Insights (cont.)



- The application insights blade provides a rich dashboard of data (tiles) and supports “drilling down” into more specific metrics.

Demonstration: Monitoring a Web Application

In this demonstration, you will learn how to:

- Create a new ASP.NET Web Application project with Application Insights referenced
- Use the Application Insights service and dashboard to monitor the metadata about your application's requests

Lesson 3: Building High Performance Applications by Using ASP.NET

- Web Hosting Performance Improvements
- Asynchronous HTTP Modules and Handlers
- The Async Keyword
- State Management

Web Hosting Performance Improvements

- ASP.NET 4.5 introduced many improvements to web hosting:
  - Asynchronous requests and responses
  - Support for Task Parallel Library and await and async keywords
  - Sharing common assemblies across applications
  - Multi-core startup (JIT)
  - Windows prefetcher for web applications

Asynchronous HTTP Modules and Handlers

- Asynchronous methods in ASP.NET allow your code that is waiting for an IO-bound operation (Database, Service, Disk) to return threads to the ThreadPool
  - Typically these requests would block the thread from servicing other requests
  - By releasing threads while waiting on external factors, the amount of requests that can be processed simultaneously increases exponentially
- Helps greatly with applications where the load spikes and is not generally consistent

The Async Keyword

- The **async** and **await** keywords are available to create easy to read and write asynchronous methods in C#
- The keywords can be used with ASP.NET MVC to create asynchronous actions:

```
public async Task<ActionResult> ItemsAsync() {
   var context = new DatabaseContext();
   var model = await context.GetModelItemsAsync();
   return View("Items", model);
}
```

State Management

- When distributing your application across multiple instances, session state needs to be shared across the instances
  - What happens when a user starts on Server1 but is now on Server2 when they click a link?
- Session state can be moved from in-memory to a dedicated session server:
  - Microsoft SQL Server
  - ASP.NET State Server
- Session state can also be partitioned and distributed among multiple session stores

Lesson 4: Common Cloud Application Patterns

- Cloud Application Patterns
- The Retry Pattern
- The Valet Key Pattern

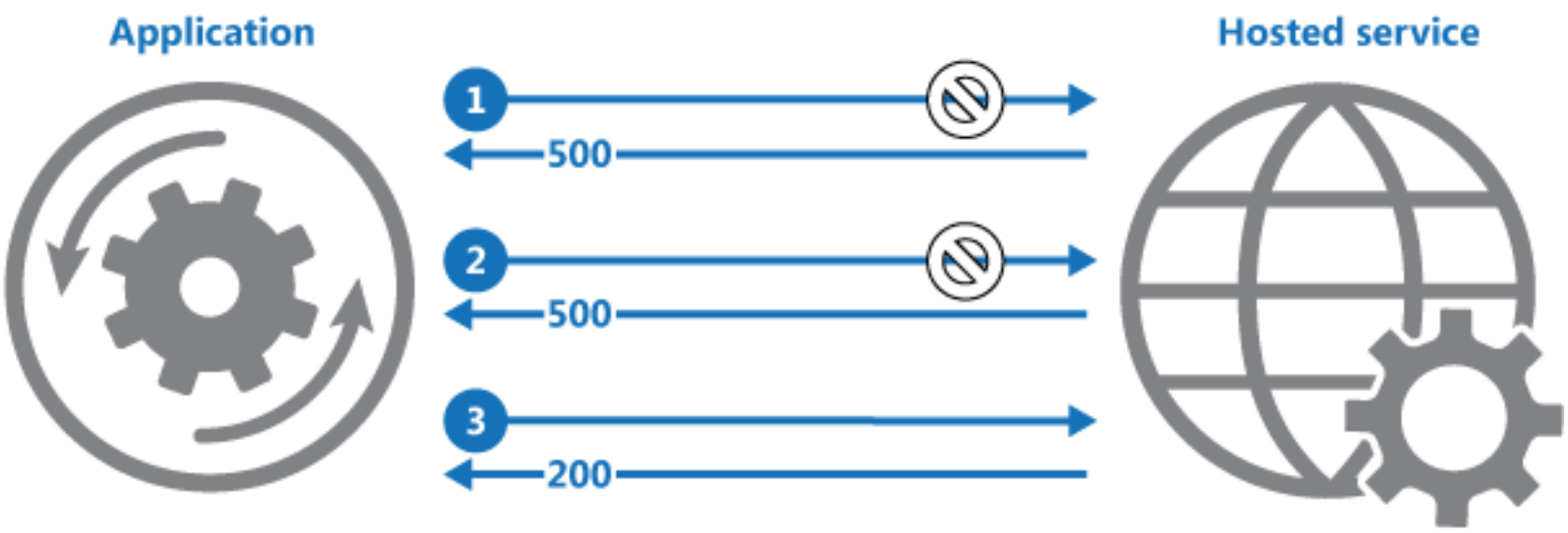
Cloud Application Patterns

- MSDN provides a collection of cloud design patterns
- Patterns:
  - Retry
  - Valet Key

The Retry Pattern

- The Retry pattern is designed to handle temporary failures
- Failures are assumed to be transient until they exceed the retry policy
- The Transient Fault Handling Block is an example of a library that is designed to implement the Retry pattern (and more)
- Entity Framework provides a built-in retry policy implementation
  - Implemented in version 6.0

The Retry Pattern (cont.)

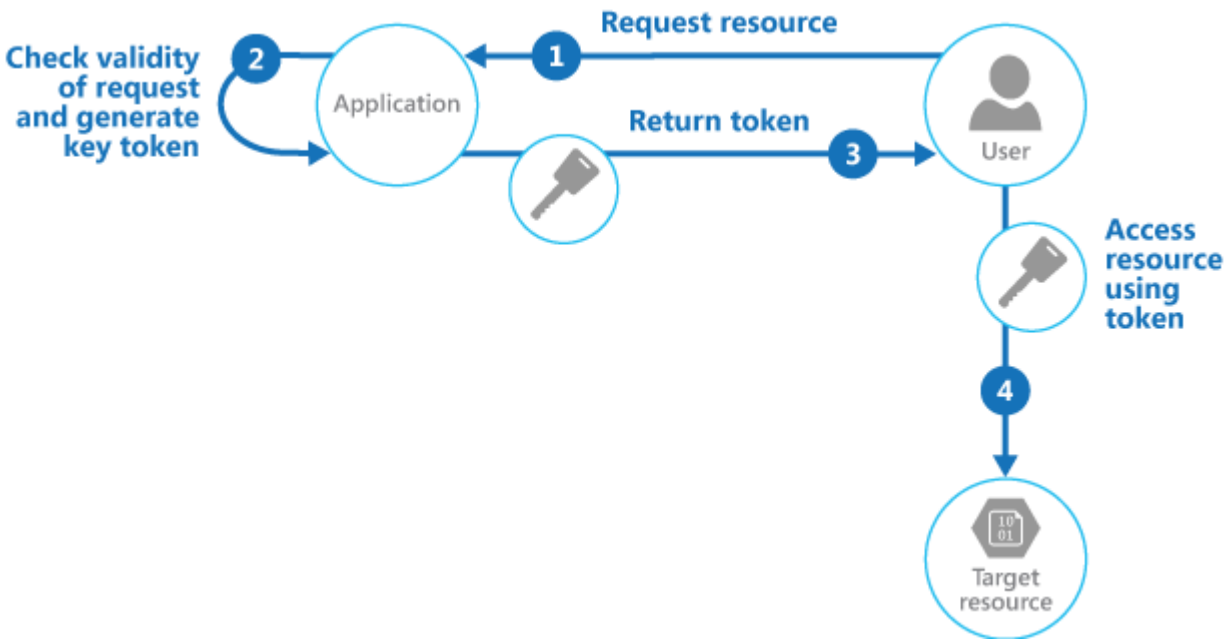


- The application sends a request to a hosted service
- The hosted service responds with a HTTP 500 (Internal Server Error) code
- The application retries until it exceeds the retry count for its policy or is met with an acceptable status code such as HTTP 200 (OK)

The Valet Key Pattern

- If your application access a resource on behalf of your clients, your servers take on additional load
- You do not want to make your resources publically available so you are typically forced to have your application validate a client
- The Valet Key pattern dictates that your application simply validates the client and then returns a token to the client. The client can then retrieves the resource directly using its own hardware and bandwidth

The Valet Key Pattern (cont.)



- The client requests a resource from your application
- Your application validates the client and then returns an access token
- The client then directly accesses the resource by using the provided token

Lesson 5: Caching Application Data

- Redis Cache

Redis Cache

- Based on the open-source Redis platform
  - Multiple tiers are available that offer different numbers of nodes
  - Supports transactions
  - Supports message aggregation using a publish subscribe model
  - Considered a key-value store where the keys can be simple or complex values
  - Massive Redis ecosystem already exists with many different clients

Module Review and Takeaways

- Review Question(s)
- References:
  - https://docs.microsoft.com/en-us/azure/app-service/choose-web-site-cloud-service-vm
  - https://docs.microsoft.com/en-us/azure/cloud-services/cloud-services-dotnet-get-started
  - https://github.com/Azure/azure-webjobs-sdk/wiki
  - https://docs.microsoft.com/en-us/azure/application-insights/
  - https://docs.microsoft.com/en-us/azure/application-insights/app-insights-snapshot-debugger
  - https://docs.microsoft.com/en-us/azure/redis-cache/cache-dotnet-how-to-use-azure-redis-cache