Enterprise Scenarios & Requirements

http://www.pluralsight.com/

Outline

- Security
- Transactions
- Reliability
- Manageability
- Scalability
- Concurrency

Security

- **Why plan for it?**
  - If you have something of value, others will try to get it
- **Confidentiality: Encrypt messages, mitigate eavesdropping attacks**
- **Integrity: Sign messages to mitigate tampering and replay attacks**
- **Authentication: Proof of identity to mitigate spoofing, impersonation attacks**

Authentication

- **Answers "who" is making the request**
- **Pick credential types you will accept**
- **Understand users before picking authentication**
- **Choices can span a wide range, but typically limited to:**
  - Anonymous
  - Username/Password
  - Windows/Kerberos
  - X.509

Authorization

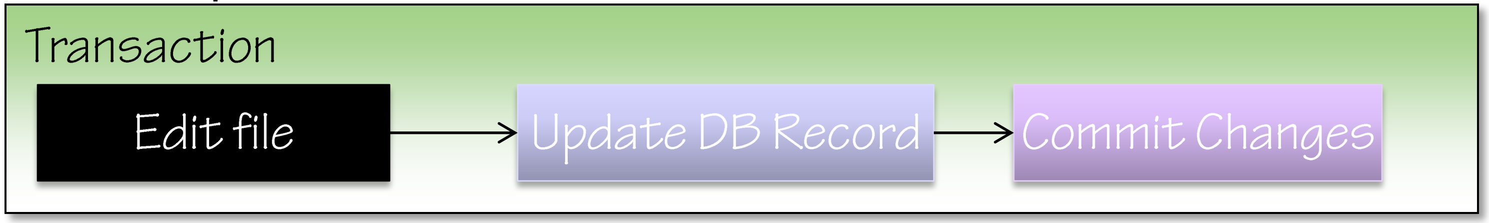
- Authorization answers “what are you allowed to do?”
- Caller makes assertion about roles it owns, includes identity.
- Callee validates that role has access to functionality
- Assertions present on Windows credentials, SAML token, OAuthWRAP token.
- Basic auth keeps mapped roles on your system. ASP.NET Role Provider handles adding roles.
- Use impersonation to delegate authorization down the stack.
  - Useful if checks happen at database, file system, or at a layer further down the stack.
- Check roles to implement authorization your self.
  - Needed if your code handles authorization
  - Use AzMan or ASP.NET role provider
- Can also check claim sets in federation scenarios. Claims signed by identity provider.

Transactions

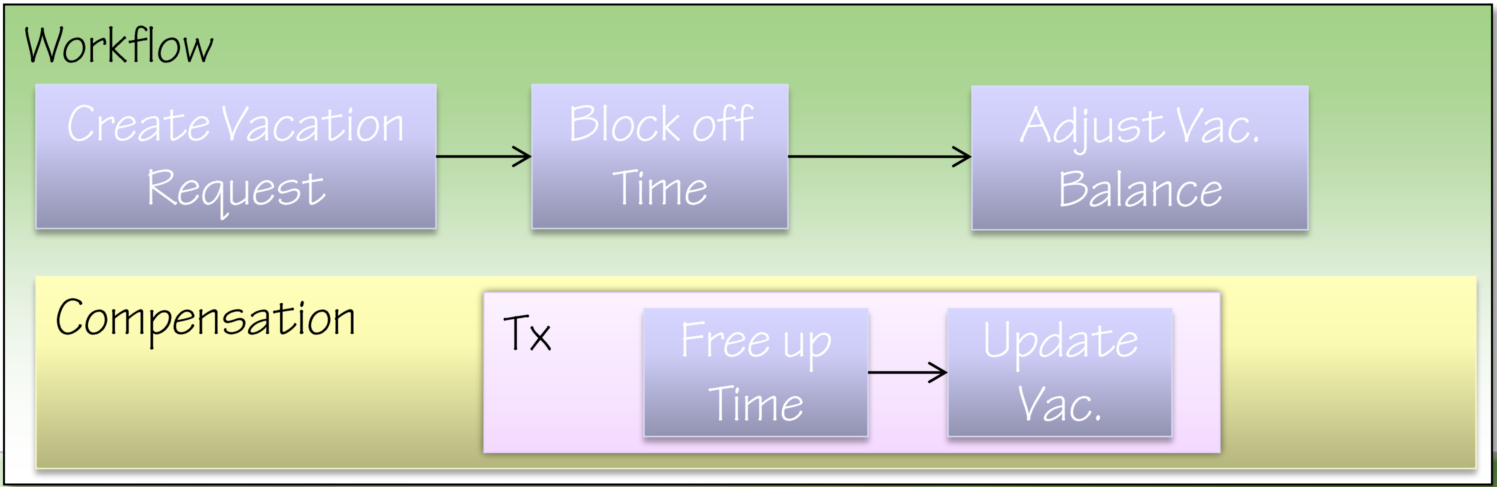
- **What is a transaction?**
  - Unit of work that completes or rolls back
  - ACID: Atomic, Consistent, Isolated, Durable
- **Players in a transaction**
  - Transaction Coordinator: Maintains ACID properties for unit of work
  - Resource Manager: Maintains previous and current state for resources involved in a unit of work
- **Transaction implementation**
  - At least one Transaction Coordinator and one or more Resource Managers
  - TC and RM communicate using XA protocol
  - Uses 2 Phase Commit:
    - Phase 1: Are you OK to commit these changes?
    - Phase 2: Commit those changes!

Options for Transactions

- When atomic work unit is a must, wrap unit of work in a transaction. Use resource managers (database, queue, file system/registry) to maintain state.
  - Example:



- When compensation is possible or work is long running, wrap work in a workflow (nicer modeling tool!)



Reliability

- **Uses: make sure my messages arrived:**
  - In order
  - Only once
- **Techniques for building reliable systems:**
  - Use WS-ReliableMessaging to send
  - Use MSMQ for message delivery (focuses availability on MSMQ machine, not receiver)
    - Use transacted queues
    - Deliver related messages to queue in a session

Manageability

- **Configuration**
    - Think about what parts of your app might need adjusting
    - Add config based on things you see people want to tweak during testing, deployment (hint: read bug reports looking for patterns)
- **Performance Counters**
    - Used to monitor the health of your application
    - Can monitor absolute quantities (X items created), rates per unit of time (changes per second), or current value (n users on line)
    - Used by admins in tools like System Center Operations Manager to trigger alerts
- **WMI Instrumentation**
    - Reveal what is happening in detail. Object model.
    - Use to tweak settings at runtime

Manageability (cont'd)

- **Event Log**
  - Used to report low frequency events (service start/stop)
  - Used to report high value events (OMG! The server is on fire!)
- **Tracing**
  - Used even in production
  - Used to report high frequency events
  - Logs need to be discoverable by IT engineering, readable by IT development
- **Event Tracing for Windows**
  - Highest performance tracing mechanism (can still kill your app performance!)
  - External interface (logman) can aggregate traces from providers to one trace file
  - Output is binary. Use tools (in Windows) to translate files to readable, XML format.

Scalability

- **Two choices for scale: up or out.**
  - ‘up’ only works if you scale well as memory increases.
  - ‘out’ takes advantage of multi-core, extra machines, etc.
- **Database techniques**
  - Use when you need relational integrity
  - Sharding is popular
    - Replicate what you must (small volume, highly used items)
    - Split what you can (high volume, less frequently used items)
- **Optimizations:**
  - For reads: Denormalization. Ex: Status updates on social media are read heavy. Implementation: NoSQL-style database, cache
  - For writes: Normalization. Ex: Shopping cart/in process purchase order. May optimize shipped orders for read.

Scalability (cont'd)

- **Caching**
  - Use database as backing store, keep information in memory
  - Examples: Windows Server AppFabric, memcached
  - Used at places like Facebook (memcached): (*see Resources)*
    - 200TB of cache
    - 2 Trillion cached items (200 million items/server)
    - Read/write: 400 million gets/s, 28 million sets/s
- **Eventual Consistency**
  - Also called BASE: **B**asically **A**vailable, **S**oft-state, **E**ventual Consistency (see Resources,)
  - Eric Brewer: CAP Theorem: Consistency, Availability, Partition Tolerance
    - Consistency: Transactions
    - Availability: The application is available.
    - Partition Tolerance: Network failures still allow the application to function
    - CAP theorem: You only get 2

Concurrency

- **Program for asynchronous usage (non-blocking)**
  - Calls out of process
  - ‘Big work’
- **Tools for concurrency:**
  - System.Threading.ThreadPool
  - Callbacks
  - Reactive Extensions for .NET (Rx)
  - Parallel Extensions (.NET 4.0)
  - Workflow Foundation

Summary

- Plan ahead for authentication/authorization by knowing what credentials your users will want to use
- Transactions guarantee consistency for short lived (<1 minute) modifications.
- Consider workflows and compensation for long lived changes (>1 minute).
- When you need to guarantee that messages arrived, use transacted queues or WS-ReliableMessaging
- Plan ahead for modifying app behavior. Add health monitoring.
- Build scalable apps through use of parallel libraries, database sharding, caching. Denormalize as needed when read heavy. Use alternative datastores.
- For concurrency- design for many CPUs, avoid blocking calls.

Resources

- XA Protocol: http://www.opengroup.org/bookstore/catalog/c193.htm
- Interpretation of data from Qcon 2010. James Hamilton: http://perspectives.mvdirona.com/2010/07/01/Velocity2010.aspx
- Cluster-Based Scalable Network Services, Fox, Gribble, Chawathe, Brewer, Gauthier: http://www.cs.berkeley.edu/~brewer/cs262b/TACC.pdf
- Reactive Extensions for .NET (Rx): http://msdn.microsoft.com/en-us/devlabs/ee794896.aspx
- CIM Studio (WMI Administrative Tools): http://www.microsoft.com/downloads/details.aspx?FamilyID=6430f853-1120-48db-8cc5-f2abdc3ed314&displaylang=en
- Mersenne Prime: http://en.wikipedia.org/wiki/Mersenne_prime

For more in-depth online developer training visit

pluralsight
see what you can learn

on-demand content from authors you trust