I have the great pleasure of participating in a 3 day course *Architecting on AWS* and wanted to share a high level understanding as a way to both share the basics for anyone “on the fence” for jumping to AWS or not, and also selfishly to help me understand and remember more clearly as I go through the course.

These are my notes from the first of three days:

## Fun facts/Anecdotes

1. NASA is planning to move ~80% of their architecture to AWS
2. AWS is a “deny-all” by default security architecture
3. Amazon leaves 100% responsibility of application layer to the “owner”
    1. Intrusion detection and security “not their problem”
    2. Determining “good” and “bad” traffic is up to the application
4. *Spot Pricing* can be up to 90% discount off standard pricing
5. Great real-time status report on AWS services: http://status.aws.amazon.com/
6. AMIs are created by region; an AMI available in US-WEST-2 will not be available in US-EAST-1 by default (for example).
7. More than 50 *Edge Locations* sitting outside of designated AZs.
    1. CloudFront and Route 53 predominantly
    2. Globally present, separate from regions
    3. Great for caching
8. AWS is completely API driven
9. Amazon.com was mandated (by CEO Jeff Bezos) that *all* development was done via APIs

## Authentication

1. Identity Access Management (IAM)
    1. Create and manage identities
    2. Not appropriate for application authentication/authorization
    3. *Federated* users (permissions for users outside of AWS)

    4. Multi-factor authentication (MFA)
        1. Available for individual IAM users, and/or AWS account
        2. Do not use master account (root) ever
2. IAM Roles
    1. Allow delegation of access to users or services that don't have static AWS credentials
    2. IAM users or AWS services can assume roles (temporarily) to make AWS API calls
3. AWS re:Invent (~November annually)
4. Interesting anecdote: You could theoretically **delete** your entire "data center" when using AWS.
5. Interacting with AWS
    1. Management Console via credentials
    2. AWS Command Line Interface (cli)
    3. Software Development Kids (SDKs)
    4. Query APIs
6. IAM Policies
    1. JSON format, deny by default, **most restrictive policy wins**
    2. If something is explicitly denied, it can **never** be allowed
    3. Evaluation logic: explicit deny/allow, implicit deny
7. Role based Access Management
    1. Assign permissions to logical and functional groups
    2. bulk permissions management (scalable)
    3. Easy to change permissions as individuals change teams
8. IAM Best Practices
    1. Remove all access/secret keys from root account
    2. Create individual IAM accounts with Admin privileges mapped to actual persons
    3. Use groups to assign permissions to IAM users
    4. Grant least privilege

# Federation

1. AWS Security Token Service (STS)
    1. lightweight web service that allows creation of temporary, limited-privilege credentials
2. Federated users
    1. authenticate users to your own identity store
        1. write an “identity broker app”
        2. users authenticate to the broker
        3. identity broker provisions temporary credentials via STS
        4. **SSO:** temporary credentials can be used to sign user directly into AWS Management Console
3. AWS Directory Service
    1. Managed Service:
        1. connect aws resources with an existing on premises Microsoft Active Directory
        2. set up a new standalone directory in the AWS cloud (simple ad)
    2. AWS Directory Service allow use of existing corporate credentials:
        1. accessing aws services (workspaces, workdocs)
        2. accessing aws management console through IAM roles
4. Web Identity Federation
    1. STS API
        1. temporary credentials to access AWS resources
    2. Supported web identity providers
        1. Amazon
        2. Google
        3. Facebook
    3. Mobile app can be developed without server-side code and without distributing long-term credentials with the mobile app