Cloud Computing Strategies

Three well-defined cloud services

* infrastructure-as-a-service
  + hosting provider provides the hardware (maybe virtualized)
  + you control everything from the OS to your apps
* platform-as-a-service
  + hosting provider provides the hardware, operating system, and usually software components (eg database)
  + hosting provider usualy handles the routing tasks like backups, load balancing, and storage management
* software-as-a-service
  + everything necessary to run app is provided
  + usually application must meet or use the hosting provider’s supported infrastructure
  + usually comes with scalability and high-availability features

Benefits of Cloud Computing

* elasticity- how well a system can adapt to changing workloads
* availability - built to be highly available by default
* on demand – requesting more resources like computing, networking, or storage is faster
* cost – paying for only what you need is usually cheaper
* don’t have to deal with challenge of running your own data center

Microsoft Azure

* Microsoft cloud computing solution provides
  + compute services – automating work
  + hosting services
  + storage services
  + database services
  + building block for services
* provides an on-demand and scalable resource platform
* can host applications in Java and other languages
* provides geo-replication for disaster recovery
* can host whole solutions or provide standalone services
* offers a set of building block services for handling identities, communication, and media
* offers features to facilitate scalability, replication, and backup

Azure Components

* PaaS offering
* Azure Storage
  + three types of storage: Blob, Table, and Queues
* Azure Service Bus
  + messaging infrastructure to exchange data between apps
* Azure Access Control Services (ACS)
  + authentication, authorization, and identity management
* Azure Caching
  + low-latency, high-throughput data store
* Azure Content Delivery Network (CDN)
  + static content caching on servers worldwide to reduce latency
* SQL Database
  + fully scalable and highly available cloud-based SQL db

Data Storage Strategies

* Common models:
  + relational
  + hierarchical
  + object oriented
* data can be stored in many formats and structures
* storage infrastructure usually had tradeoffs between latency and capacity, and how it can be queried
* File System
  + stores large amounts of unstructured data
  + uses Directories and Files to store and manage data
* Distributed File System (DFS)
  + data distributed to many systems and disks
  + user sees just a single disk
* Relational Databases (RDBS)
  + designed to store large amounts of structured data and handle complex queries
  + uses language like SQL to execute CRUD operations
* NoSQL Database
  + designed to store large amounts of unstructured data in key-value pairs, documents, or graphs
  + generally, they are schemaless
* In-memory Stores
  + very limited storage of temporary data in memory
  + not persistent but very fast
* Distributed Caches
  + reduces latency by caching data
  + uses multiple computers to store cache in memory
  + seems like a single local cache to user

Hypertext Transfer Protocol HTTP

* is a communication protocol
* an application layer protocol
* the primary protocol for most web applications and web services
* is widely supported, easy to scale, and easy to use
* uses the request-response message pattern
* HTTP uses an address to request resources
* request contains-
  + HTTP method
  + request URI (address for the request)
  + HTTP version
  + Headers
    - used to provide metadata to the server
    - content negotiation
* response contains
  + status line with
    - HTTP version
    - status code
    - reason phrase (description of status coe)
  + headers
    - metadata for the server and response
    - contains “Content-Type” and “Content-Length” for optional body
  + body (optional)
    - contains the requested data
* Uniform Resource Identifiers (URI) is the address for an HTTP request
  + ex <http://mysite.com>:8080/mypage.html?section=1
  + http:// - protocol specification
  + mysite.com – hostname of the server
  + :8080 – optional port (default is 80)
  + /mypage.html – optional path to the resource
  + ?section=1 – optional query details to describe or define the resource request
* HTTP requests use verbs to declare an action
  + 8 defined verbs
    - GET, HEAD, OPTIONS, POST, PUT, DELETE, TRACE, CONNECT
  + most are considered safe meaning they should have no effect on resource state
  + most verbs are idempotent meaning they can be sent to server multiple times and the state should be the same for each
  + when sending a request, the verb is in the first part of the request line (HTTP method)
* Media types
  + HTTP was designed for HTML but other resource types need to be transferred (json, etc)
  + to support different data or media types HTTP uses the multipurpose Internet mail extensions (MIME) types
  + Content-Type field in HTTP header sets the content type
    - examples are text/html, text/plain, application/json
* Hypermedia
  + differentiates REST from HTTP services
  + allows clients to dynamically find capabilities provided by a server or service
  + does this by returning hypermedia artifacts
    - the artifacts provide details to the client on the actions that can be performed
  + media types like XHTML and ATOM support some hypermedia artifacts already

Cloud Storage

* Personal Cloud Storage
* Public Cloud Storage
* Private Cloud Storage – stored in the enterprise’s data center
* Hybrid Cloud Storage – combination of public and private
* Advantages
  + file accessibility
  + off-site backups
  + less local storage is cheaper
* disadvantages
  + data ownership
  + bandwidth constrictions
* Cloud Communication – internet based voice communications which is hosted by a third party outside the organization and is accessed via the internet

Azure Stuff

* Search Services
  + Azure Search Services
    - fully managed cloud service
    - offers search capability
    - includes natural language support
    - provides scalability
    - is API based
* Redundancies in Azure
  + locally redundant (saved in a server in same geographic region)
  + geo redundant (save in different region)
  + read-access geo redundant
* Fault Tolerance
  + fault domain
  + automatically controlled
  + 100% uptime
* Windows Server Active Directory
  + Active Directory Domains and Trusts – to configure domain-specific information
  + Active Directory Sites and Services allows us to create structures
  + Active Directory Users and Computers then allows us to define, change, or delete objects within Active Directory
* Company Portal in Intune
  + allows connection of device to Microsoft Intune
  + allows for downloads of company applications
  + allows a company to apply settings, collect information, install, and remove apps
* Work Folders
  + single point of access
  + allows you to work offline
  + maintains encryption
  + use existing technologies

AWS EC2 Overview

* windows and linux virtual machines (instances)
* Amazon Machine Image (AMI)
  + templates for instances
* configurations of CPU, memory, storage, network, instance types
* secure login using key pairs
  + mathematically related public and private keys
  + generated by EC2 or import your own
  + public key is stored with Amazon
  + private key with you and used when connecting to the instance
* Linux- SSH
* Windows – RDP Remote Desktop Protocol
* Firewall
  + each instance can be associate with a security group
  + security groups acts as a firewall controlling traffic
* Virtual Private Cloud (VPC)
  + default VPC or customizable
  + isolated from other VPCs
  + customizable subnet ranges, network ACLs, etc
* Elastic IP address (EIP)
  + static public IP address associate with AWS account
  + gets mapped to a VM instance
  + can be remapped to other instances
  + limit of 5 EIPs per VPC
* Elastic Block Store (EBS)
  + persistent storage
  + attached to instances
  + behaves similar to a physical hard drive
  + can be encrypted
* Instance Store
  + temporary storage for an instance while the instance runs
* Simple Storage Service (S3)
  + cloud data storage
* tagging – add meaningful metadata in form of key/value pairs

Amazon Machine Image

* used to create EC2 virtual machines
* includes
  + a template for the root volume
  + launch permissions specifying accounts that can use the AMI
  + block device mappings for EBS volumes
* AMIs can be made public or shared with specific account
* bought and sold on AWS marketplace
* can be used to create multiple EC2 instances (one-to-many)

Accessing EC2 instances

* OS-level access requires different credentials from AWS
* you own the OS credentials
* you must authenticate to an instance at OS level
* AWS provides asymmetric RSA key pairs
* Windows uses ec2config service
  + ec2config sets random (encrypted) admin password
  + user can get the password with the private key

EC2 Security Groups

* virtual firewall for EC2 instances
  + create a security group
  + assign rules
  + modified rules apply to all group members
* VPC
  + isolated customer network
  + can change rules on a running instance
  + 5 groups per instance and 50 rules per group
* security group rules control inbound and outbound traffic
* rules specify
  + protocol (UDP, TCP, or ICMP)
  + port
  + ICMP type
  + source and destination – IP range or security group
* custom security groups require a name and description
* default rules allow no inbound, allow all outbound

EC2 Snapshots

* applies to EBS volumes
* supplements existing backup solution
* can be used to create a baseline
  + new volumes can be created from the snapshot
* they pick up only incremental changes
* encrypted EBS volumes
  + snapshots also encrypted
* unencrypted snapshots can be shared with other AWS accounts or public
* root device EBS volumes
  + stop the instance or unmount

CloudWatch Monitoring

* monitor AWS resources
* track performance metrics
* log file collection
* set alarm notification
* EC2 Monitoring
  + basic (free)
    - pre-selected metrics
    - 5 minute frequency
  + detailed
    - all metrics available
    - 1 minute frequency
* aggregated metrics
  + auto scaling
  + load balancing
* 2-week retention period for monitoring data
* CloudWatch dashboards

AWS Identity and Access Management (IAM)

* identity and access management
* shared access to AWS account
* granular permissions
* secure resource access
* multifactor authentication
* identity federation
  + integrates with most AWS services
  + free
  + providers-
    - Amazon
    - Facebook
    - Google
    - OpenID Connect
* access IAM through
  + AWS management console
  + AWS command line
  + AWS SDK
  + IAM HTTPS API
* IAM user is an identity or application
* users can’t access anything by default
* attach policy to IAM group to give permission
* policies are attached to a user (user-based) or resource based
* resource-based policies specify the Amazon Resource Name ARN
* AWS account
  + used to sign up and represent business relationshiop
  + has root permissions to all resources
  + an organization can have several
* IAM users
  + multiple IAM users under a single AWS account
  + can be a person or application
  + fine-grained permissions

Active Directory Integration

* create Microsoft Active Directory in AWS
  + Simple AD
* connect your existing on-premises MAD to AWS
  + AD connector
* Virtual Private Cloud (VPC)
  + 2 VPC Subnets in 2 different availability zones
* connection between customer network and VPC
  + hardware VPN – IPsec VPN tunnel
  + software VPN – VPN appliance
  + AWS Direct Connect – dedicated 1 or 10 gbps
* account used by AD Connector
* IPs of 2 on-premises DNS servers
* TCP/UDP ports

1. user signs in through access URL over SSL
2. AWS uses LDAP authentication to on-premises Active Directory
3. user accesses AWS services

Elastic Container Registry ECR

* Amazon EC2 Container Registry 🡪 Repositories 🡪 Get started
* create repository for customization and better security
* $ aws ecr get-login –region us-east-1
* ^ code used to reeive the docker login command needed to connect to your registry
* $ docker build –t demo .
* $ docker tag demo:latest asdfjlk
* $ docker push asdfjlk
* Create Cluster
* Amazon EC2 Container Registry -> Repositories -> Get started

Task Definition

* task definition is a JSON-formatted document that specifies how to deploy one or more Docker containers that build up an application
* its what is used by Amazon ECS to launch both tasks and services
* has name, links, image, essential, portMappings, memory, cpu, logConfiguration, command

OpsWorks and Lifecycle Events

* setting up a sample stack
  + create a stack
  + set the repository of the stack
  + create a layer named node.js app server
  + assigning a recipe to the deploy lifecycle event
  + adding an instance to the layer
* each layer is a collection of EC2 instances which share a configuration
* OpsWorks lets you manage applications and servers

Cloud Computing

* measured service – use what you use
* on-demand self-provisioning
* resource pooling
* rapid elasticity
* broad network access
* Public Cloud
  + accessible by Internet
  + cloud provides hardware
* Private Cloud
  + accessible only to single organization
  + organization owns hardware infrastructure
  + still adheres to cloud characteristics
* Hybrid Cloud
  + on-premises and cloud
  + migration of on-premises systems and data takes time
* Community Cloud
  + same needs across multiple tenants
* Hosted IT services
  + servers, storage, databases, and web apps
  + managed services
  + pay for services used
  + self-provisioning
* Shared responsibility
  + degree depends on specific cloud service
  + more responsibility means more control
    - subscriber responsibility
      * users and groups
      * data
      * security
      * maybe software
    - provider responsibility
      * security
      * hardware
      * maybe software
* Cloud Computing Service Models
  + IaaS - Infrastructure
    - EC2 virtual machines
    - S3 storage
    - VPCs
      * virtual network
  + PaaS - Platform
    - databases
    - Amazon CloudSearch
  + SaaS – Software
    - web sites
    - WorkDocs
  + SECaaS - Security
    - AWS Shield
      * DDoS protection
* Data Centers
  + physical facilities
    - house thousands of servers
    - many locked racks full of equipment
    - HVAC
    - power generators
    - multiple internet connections
  + security
* Regions is a collection of one or more Availability Zones
  + geographic area
  + new AWS resources can be placed in specific regions
  + some AWS service are unavailable in some regions
  + AWS resources are not replicated across regions by default
  + there is a charge for cross-region data transfers
* Availability Zone AZ
  + isolated locations within a region
    - eg us-west-1a
  + AZs are linked together with high-speed links within a region
  + an AZ is not necessarily a physical data center
    - larger AZs have multiple data centers

Networking Components

* VPC – Virtual private cloud
  + isolated network in the cloud
  + custom IP address range
  + network gateways
  + contains subnets
  + routing table configuration
* subnets – smaller network within a larger network that increases performance
* VPN – Virtual Private Network a network we define in the cloud
* ELB – Elastic Load Balancer
  + improves web application response time
  + incoming client request get routed to backend instances
  + traffic not sent to unresponsive backend nodes
  + works across AZs
  + autoscales
  + class load balancer
    - routes traffic based on network address and port info
  + application load balancer
    - routes traffic by looking at application specific request info in the request content
* Direct Connect – dedicated private network link to AWS cloud
  + does not use internet
  + predictable network throughput
  + increased security
  + 1 and 10 Gbps
* Route 53 – DNS name resolution solutions
  + DNS – domain name system, DNS traffic uses port 53
  + registers DNS domain names
* CloudFront – content delivery network CDN
  + speeds up web site content delivery to users
  + client requests get routed to the nearest content location
  + works with static and dynamic web site content

AWS EC2 Overview

* Windows and Linux virtual machines (instances)
* Amazon Machine Image (AMI)
* instance type
  + CPU
  + memory
  + storage
  + network
* secure login using key pairs
  + mathematically related pubic and private key
  + generated by EC2 or imported

Container Basics

* isolated app runtime environment
* the OS and some binary libraries are shared by containers
* containers are based on images
  + images contain software and settings for running container
  + images contain metadata describing the image
  + a container is a runtime instance of an image
* necessary elements-
  + software
  + settings
  + app-specific libraries
  + runtime environment
  + tools
* data persistence
  + application writes persist until container is deleted
* container scalability
  + address peak application requests
  + try not to run multiple services in same container
* container versioning
  + only incremental changes are read

Dev Tools

* AWS CodeCommit
  + git repository source code control (central)
* AWS CodePipeline
  + a change in code triggers- Build, Test, Deploy
* AWS CodeBuild
  + compiles code
  + produces software packages
* AWS CodeDeploy
  + automates deployment of code
    - EC2 instances
    - on-premise servers
    - updates
* Cloud High Performance Computing HPC
  + AWS managed service
  + used for complex computer scenarios
  + pay for what you use
  + EC2 compute power for processing parallel tasks
    - vertical scaling with increased vCPUs
    - horizontal scaling with addition of cluster nodes through Elastic HPC Cluster
  + disk performance
  + network performance
  + standard AWS security can protect HPC workloads

Cloud Storage

* Amazon Simple Storage Service S3
  + cloud objects storage
  + supports trillions of files
  + supports hundreds of thousands of transactions/second
* S3 storage buckets
  + files get stored in bucket folders
  + deploy the bucket in a region where access is required (to reduce latency)
  + bucket names must be unique
  + bucket endpoint URL <http://bucket1.s3.amazonaws.com>
  + private by default
    - access can be granted though a policy
      * policies can apply to objects, buckets, or users
  + S3 replication
    - 3 availability zones by default
    - cross-region replication available
  + S3 security
    - TLS encryption of data in transit
    - default encryption can be enabled for data at rest
* S3 Management
  + $ New-S3Bucket $ Get-S3BucketLocation $ Remove-S3 Bucket
  + powershell cmdlets ^
  + AWS management console GUI
  + AWS CLI
  + APi programmatic access

Amazon Glacier

* low cost archiving service
  + begins with creating a vault
* designed for use with other Amazon web services
  + infrequently used S3 data can be oved into Glacier through S3 lifecycle policies
* archive retrieval initiates a job
  + data is not immediately available
  + retrieval jobs can take up to 5 hours
  + specify the range of retrieval
* Archives
  + single file
  + .tar or .zip file
  + improve experience by uploading parts separately
* data is stored in multiple facilities for greater durability
* regular data integrity checks
  + automatic self-healing
* vault permissions
  + initiate job
  + list jobs
  + upload archive
* data at rest protected using server-side encryption
* monitor data access using AWS CloudTrail
* pay only for what you use
  + no minimum fee
* 1st step is to configure vault
  + name
  + event notifications
* $ aws glacier upload-archive –account-id 1324 –vault-name etc

AWS storage solutions

* Amazon Elastic File System EFS
  + shared folder in the cloud for NFS connectivity
* Amazon Elastic Block Store
  + EBS volumes are used by EC2 instances
  + can attach and detach them to selected virtual machines
* Amazon EC2 Instance Storage
  + used by some EC2 instances automatically
    - depends on instance type
    - specified when an instance image is created or launched
    - will show up as an ephemeral (temporary) volume type
  + provides temporary block-level storage for an EC2 instance
    - use EBS volumes for frequently accessed persistent data
    - aside file level storage – most commonly used for storage systems, files are stored and accessed in bulk
    - block level storage – raw volumes of storage are created and each block can be controlled as an individual hard drive, blocks are controlled by server based OS and each block can be individually formatted
  + stored on disks physically attached to host computer in an AWS data center
  + ideal for temporary storage of information
  + not a durable disk storage type
    - data persists during the life of the associated EC2 instance
    - data is persistent across instance reboots
* AWS Storage Gateway
  + provides AWS storage to on-premises IT services
    - emulates NFS, iSCSI file server
  + requires an on-premises AWS Storage Gateway appliance
  + cloud-stored data is cached on the applicate for quick on-premises access
  + initial configuration –
    - connect to IP address of the Storage Gateway appliance once it is running, to activate the gateway virtual machine
    - IP address must be accessible to the web browser
  + Storage Gateway Types
    - file gateway
      * AWS S3 object storage via NFS
    - volume gateway
      * AWS S3 block storage with EBS snapshots
    - tape gateway
      * AWS S3 storage through existing tape backup solution, long-term storage in Glacier can be configured
* AWS Snowball
  + AWS storage offering
  + can move very large amounts of data into or out of the AWS cloud
  + cheaper and quicker than transferring data over the internet
  + one can store TBs of data
  + Snowball devices use Trusted Platform Module (TPM) for data integrity and encryption
  + AWS Snowball Process
    - create a Snowball job
    - the job tirggers a Snowball device being shipped to your location
      * attach to your on-premises network
    - install the Snowball client
      * specify data to be transferred to appliance
      * data is encrypted on the appliance
    - after job completes a shipping label is generated
* AWS CloudFront CDN
  + content delivery network
  + delivers in a nearby region
  + 1. create distribution

Loose Coupling

* isolation
  + application tiers
  + application components
* design considerations
  + component isolation
    - tiers
    - VPCs virtual private cloud
    - containers
  + component messaging
    - sync
    - async
* sync decoupling
  + components must be available at the same time
* Amazon Simple Queue Service SQS
  + queue – temporary storage location for messages
  + 1 to 14 days, default is 4
  + delivery is guaranteed even with intermittent network connectivity
  + queues scale automatically
  + is a managed service
  + unlimited number of messages
  + queue messages can be encrypted
  + queue type must be chosen upon creation
    - standard queues
      * high scalability
      * near-FIFO
    - FIFO queues
      * preserver message ordering
  + visibility timeout – time when other components are not allowed to retrieve the same message
  + enables decoupling of applications
  + highly available and reliable
* Amazon Simple Workflow Server SWF
  + cloud service offering
  + coordinates execution of tasks
  + dev tool for building and coordinating loosely coupled app components
  + handles task dependencies, scheduling, and concurrency
  + can write code in multiple languages
  + Tasks – steps to run a business process within an app
  + Worker - a program that executes a task
  + Decider – coordinates tas
* Amazon Simple Notification Service SNS
  + fully managed cloud service
  + coordinates messaging to subscriber endpoints
  + manages service for mass message delivery
    - normally used for internal application messages
  + scalable web service that supports multiple subscriber types
  + SNS Topic –
    - message recipients subscribe to a topic
  + configuration and management
    - AWS management console
    - AWS CLI
    - PowerShell
    - AWS SDK

Architecting for the Cloud

* High Availability HA
  + storage
  + ec2 instances
  + managed services such as RDS
* Service Level Agreement SLA
  + uptime
  + expected performance
  + consequences
* Single Points of Failure--
* Elastic Load Balancing ELB
  + multiple back-end EC2 instance
  + auto scaling to replaced failed instances
* RDS Relational Database Service
  + multi availability zone deployments
  + read replicas
* SQL Server standby replicas
  + uses sync replication
  + automatic promotion
* replication across AWS regions
  + s3 buckets
* EBS volumes
  + automatically replicated within the AZ
* do not run multiple critical workloads in the same AZ
* DNS
  + configure DNS per region if using custom DNS servers
  + Route 53 is highly available managed service
* Five Pillars

1. Operational Excellence
2. Security
3. Reliability
4. Performance Efficiency
5. Cost Optimization

Multi-Tier Architectures

* app or system consisting of multiple decoupled components
* Amazon API Gateway
  + creation and management of APIs
  + links tiers via HTTPS API requests
* AWS Lambda
  + runs code in the cloud (business logic)
  + managed service (no need to manually provision servers)
* benefits
  + loose coupling
  + tier management may not affect other tiers
* each tier can exist in a separate VPC
  + Tier 1 : front–end
  + Tier 2 : business logic
  + Tier 3 : back-end or data tier
* Mobile Apps
  + design considerations
    - user authentication
    - push notifications
    - track app usage
      * Amazon Mobile Analytics
    - back-end data stores
    - offline functionality
    - sync when online
      * Amazon Cognito Sync
  + multi-factor authentication MFA app sign-in
    - Amazon Cognito
      * mobile app management
      * identity store
  + Amazon Mobile Hub
    - integrate AWS services into a mobile app
      * a cloud configuration file is generated
        + JSON file regarding available back-end resources
      * the AWS Mobile SDK gets added to mobile app
      * mobile app can then make API calls to AWS services
        + Amazon API Gateway custom REST APIs can talk to back-end services
* Microservices Environment
  + Amazon Elastic Container Service
    - one or more ms can run within a container
  + AWS Lambda
    - code provisioning without server provisioning
  + Amazon Simple Queue Service
    - decoupling due to stored ms messages

Cloud Application Design

* Cloud Design Principles
  + testing
    - rapid elasticity
    - VPC
      * network isolation (sandboxing)
      * rapid resource deprovisioning upon testing completion
  + recovery
    - short term data storage
      * s3
    - long term
      * glacier
    - AWS Storage Gateway
    - EBS volume snapshots
    - load balancing
    - clustering
  + scaling
    - horizontal – add/remove compute nodes
    - vertical – increase/decrease compute resources
  + automation
    - scriptable
    - remove human error
    - AWS Systems Manager
      * automate deployment and management tasks related to EC2 instances and other resources
    - CloudFormation templates
      * repeatable resource deployment
* High availability
  + continuous access to deploy resources
  + SLA
  + consider on-premises issues
    - redundant network links
    - reliability of end-user computing devices
* Placement of cloud resources
  + Availability Zones
  + Regions
  + RDS read replicas
* application load balancing with autoscaling
* multi-AZ RDS deployment uses synchronous replication so that all of the read replicas have saved the data before the update is finished (only 1 needed for asynch)
* consider resource limits
  + network throughput
  + storage
  + compute
  + concurrent application database read/writes
* Regions
  + geographical physical regions around the world
  + contain one or more AZs
  + low latency, high throughput, high redundancy within zones
* Availability Zones
  + highly available, fault tolerant, and scalable
  + high speed fiber optic networking ensures automatic failover without interruption between AZs
* End Points
  + URLs

CloudFront

* Web app content caching
  + dynamic – generated per request
  + static – unchanging
* data is replicated to different geographic regions
  + CloudFront edge servers
* network latency is reduced for user requests
* static content delivery acceleration
* CloudFront Distributions
  + reduces web app network latency
  + copies origin content to global CloudFront edge locations
  + increases web app performance and user experience
* web distributions
  + dynamic and static content
  + live streaming
  + media file access via HTTP or HTTPS
* RTMP distributions
  + Real-Time Messaging Protocol
  + optimized for delivering streaming audio and video over the net
  + media files must be stored in S3 bucket
* settings
  + origin domain name/path
    - s3, external web server, AWS MediaPackage or AWS MediaStore endpoint
  + cache behavior
    - HTTP to HTTPS redirection
    - Time-To-Live TTL – default 24 hours 86,400 seconds
  + AWS Lambda function association based on event
    - view request/response, origin response
  + default root object
    - default CloudFront object when users connect to CF URL
  + logging – user requests for distributed content
* create --- Web or RTMP distribution
* CloudFront security
  + HTTPS
    - end-to-end network protection
    - uses SSL/TLS
    - HTTP to HTTPS redirection
  + S3 origin content
    - force access indirectly via CF URLs instead directly through S3 URLs
  + restrict access to cached objects with signed cookies or URLs
    - web app must be designed to use these
    - signed URL valid date and time
    - signed URL expiration date and time
    - IP addresses that can access content
  + asymmetric encryption (public/private key pair)
    - configure CF with a public key
    - specify which HTTP POST fields should be encrypted
      * useful when users upload sensitive data
      * 10 fields max per HTTP request
    - occurs at the CF edge nearest the user
    - other app components accessing encrypted fields
      * they must have private key
    - 1. acquire an RSA public/private key pair
      * OpenSSL, AWS Certificate Manager, Microsfot Certificate Services
    - 2. configure a field-level encryption profile
      * specify which fields need encryption
    - 3. configure a field-level encryption configuration
      * HTTP content type/query parameters get matched to a config
    - 4. cache behavior
      * the config must be associated to a CF distr cache behavior

Route 53

* an AWS service that provides name resolution for DNS, Domain Named System
* Fully Qualified Domain Name FQDN
* resolves FQDNs to IP addresses
* hosted DNS zone support
  + specify a DNS domain name
* hosted DNS zone types
  + public hosted zone
  + private hosted zone for Amazon VPC
    - specify the VPC
* record sets
  + contains DNS resource records RRs such as
    - A, AAAA, MX
  + can be imported from a DNS zone file if available
* health checks
  + run against AWS endpoints, such as EC2 instance on a timed basis
* traffic policies
  + determines how Route 53 responds to DNS queries
  + visual editor tool can be used
    - traffic policies can be imported (in JSON)
    - new traffic policies can be created
  + versioning allows roll back to previous traffic policies
    - traffic weighted rule – how much traffic to take
      * 0 disables routing
      * relative numeric values
      * weights of 1 and 4 means 4 gets ¾ of requests
    - failover rule
      * provides DNS failover
      * based on health checks
    - geoproximity rule
      * based on physical location
    - latency rule
      * lowest network latency between the DNS client and target region is chosen
    - multivalue answer rule
      * returns a list of matching records when DNS records have same name and type (but different IPs)
      * not as effective as a true load balancer
* DNS Resource Records – RR
  + types of DNS records in a record set
  + responses vary depending on RR type
* caller reference parameter – random string that uniquely identifies the zone creation request

Cloud Databases

* Relational
  + structured schema
    - specific field (column) definitions
  + db tables can be linked together
    - unique foreign key
    - one or more column values
  + all rows (records) store the same type of data
  + SQL deployed within an EC2 instance
    - the most flexibility
    - more responsibility
    - configuration, maintenance, troubleshooting
  + Amazon RDS
    - less flexibility
    - less configuration, maintenance, troubleshooting
    - “managed” cloud service
    - 6 db engines
      * Amazon Aurora
        + distributed db, up to 15 read replicas
        + MySQL/PostgreSQL compatible
        + auto-scaling up to 64 TB per db instance
      * Oracle
      * Microsoft SQL Server
      * PostgreSQL
      * MySQL
      * MariaDB
* NoSQL
  + not a relational db
  + no fixed data schema
    - no specific field definitions
    - each field can store completely different types of data
  + useful when working with big data
  + increased performance through scaling out (adding more nodes)
  + partition keys are used to locate data
  + common NoSQL db types
    - document – semistructured XML or JSON document storage
    - columnar – column reads and writes as opposed to rows
    - memory key-values - cached items for read-intensive applications
  + NoSQL terms
    - table (SQL), document collection
    - document, row (SQL), item
    - field, attribute = column (SQL)
    - primary key (SQL), object ID
  + Amazon DynamoDB
    - Fully managed cloud NoSQL database
* Data Warehouse
  + complex query execution across big data
  + supports a multitude of source data types
    - S3
    - Amazon RDS
    - o-p data sources can be streamed to Redshift using AWS Database Migration Service DMS
  + Amazon Redshift data warehouse
    - fully managed cloud service and analysis solution
    - purpose – to provide quick answers to complex business questions
    - fast performing analysis queries
      * result set caching
      * high performance compute hardware
      * parallel processing
    - Database encryption
      * KMS or HSM
    - Redshift clusters
      * cluster node types
      * resizable
        + GUI
        + APIs
      * define a VPC for the Redshift cluster
        + set public visibility option
    - Redshift Spectrum
      * S3 unstructured data querying directly in S3
      * directly run SQL queries against exabytes of unstructured data
      * fast query performance
      * columnar storage technology
        + improves I/O efficiency
        + parallelize queries across multiple nodes
    - Redshift leader node
      * client queries sent here
      * coordinates query parallel execution across the cluster
      * collects results from compute nodes (non-leader nodes), sends to client
* In-Memory Data Stores
  + RAM is many times faster than disk access
  + commonly used to increase database read performance
    - frequently accessed data is served from the memory cache
  + data persistence
    - transaction logs
    - disk volume or database snapshots
  + consider the required amount of RAM
    - workload
    - data types
    - if clustering will be used
  + Amazon ElastiCache
    - Redis or Memcached
    - specify the number of replicas/nodes
    - supports multi-AZ deployments
* Graph Databases
  + focuses on the relationship between data
  + does not use a predefined schema like relational dbs
  + designed to execute complex queries quickly
    - data is organized into relationships
  + Amazon Neptune
  + nodes
    - graph entities containing attributes
  + relationships
    - identifies connections between nodes

AWS S3

* Buckets
  + identify where an object belongs
  + all s3 objects reside within a bucket
  + buckets are a flat object store
  + ex- bucketName/prefix/objectName.jpg
* prefix
  + name or characters before an object name
  + ends with forward slash /
  + groups objects together with the same prefix in their key name
  + ‘Create Folder’ option in the console creates a 0-byte named prefix with /
* objects
  + files
  + logical entities or containers with information used to store data on S3
  + can contain any data, usually bytes in specific order
* key
  + unique name and path to an object in S3
  + excludes the bucket name
    - /prefix/objectName.jpg
* value
  + the literal value or characters that correspond to a key
  + key value pair of- “name:Tom”
* version ID
  + store multiple copies of an object using same object name
  + each copy has unique version ID
  + objects uploaded before turning on versioning will not have version ID
* metadata
  + key-value pairs that describe the object
  + default - includes last modified date
  + standard HTTP metadata includes object content type
  + custom detadata can be specified when object is stored
  + view MD5, which is checksum of an object, to verify integrity
* access control (ACLs)
  + originally used to control access to objects before AWS IAM
  + Identity and Access Management now standard
* storage classes
  + each object in S3 is associated with a storage class
  + different S3 storage classes provide option to save money
  + S3 Standard
    - original and most popular class
    - preferred for scenarios where you anticipate you will be accessing an object often (hot workloads)
    - replicates data across three different physical Availability Zones
  + S3 Intelligent-Tiering
    - automatically shifts objects in and out of standard and standard-ia classes
    - good for unknown request rates
  + S3 Standard-IA
    - meant for infrequently accessed objects
  + S3 One Zone-IA
    - best for light workloads
    - in only 1 AZ
  + S3 Glacier/ S3 Glacier Deep Archive
    - designed for cold workloads (infrequent)
    - not available for real-time access
* S3 Features
  + Versioning
    - keep multiple copies of an object
    - once enabled, every new object gets version ID
  + Server access logging
    - provides detailed records for the requests made to a bucket
    - useful for reviewing access log information for security and access audits
  + Static website hosting
    - allows static website hosting on S3
    - like web pages and some client-side scripts
    - not server-side scripting
  + Object-level logging
    - allows AWS CloudTrail to log data events for objects in S3 bucket
    - CloudTrail supports logging by object-level API operations, also known as data events
    - CloudTrail can log data events for object in an S3 bucket
  + Default Encryption
    - automatically encrypts new objects with selected encryption type
    - choose S3 server-side encryption, AWS managed encryption key, or an AWS customer managed encryption key (CMK)
    - ensures that the customer’s data is encrypted at rest
  + Advanced Settings
    - Object Lock, Tags, Transfer Acceleration, Events, Requester Pays
* S3 Management
  + Lifecycle
    - automatically transition objects into a less expensive storage class
    - auto deltee and expire objects after certain number of days
  + Replication
    - auto replicate all new objects added to a bucket (to another bucket)
    - objects already in bucket not eligible
  + Analytics
  + Metrics
  + Inventory
* S3 Select
  + the select feature can run SQL queries in place against CSV and JSON objects
  + not necessary to download an object and then query
  + it is possible to offload compute resources to S3
* Access
  + public access is granted through access control lists, bucket policies, access point policies, or a combo
  + can enable ‘block all public access’
  + settings apply account-wide for all current and future buckets and access points
  + confirm apps will work correctly without public access
* Limits
  + S3 buckets do not have hard limits for the number of requests
  + no limit to amount of objects or data you can put into a bucket
  + bucket names must be DNS compliant and 3-63 characters long
  + default number of buckets per account is 100
    - hard limit of 1000
* Basics
  + object storage
    - you can upload files to a web server
    - files are split into smaller bits of data
    - data is replicated across hosts in distributed architecture
  + common use cases
    - data analytics
    - media hosting
    - backup and disaster recovery
    - static website hosting

AWS Caching

* keeps recently accessed data available
  + close geographic location
    - reduces network latency
  + memory cache
    - quicker than disk access
* CloudFront CDN
  + configure settings to replicate data to geographic edge locations
  + TTL Time To Live
    - set how long cached data is refreshed from the origin
    - default 24 hours
* AWS Storage Gateway
  + virtual machine with storage
  + allows on-premises applications to access cloud storage
    - data can be cached on the appliance for quick access
* ElastiCache
  + in-memory database cache
    - placed between an app and the back end data store
    - the memory cache is always checked first
  + greatly improves performance for read-intensive apps
  + choice of engines
    - redis
    - memcached
* DynamoDB Accelerator (DAX)
  + in-memory cache for DynamoDB (NoSQL)
* AWS Greengrass
  + software used to extend AWS cloud capabilities to customer edge devices
  + an extension to AWS IoT (Internet of Things)
  + edge devices can gather data and remain functional offline
  + customers can use event-driven Lambda functions to build app logic for IoT devices
    - lambda functions can also talk to AWS services
    - normally written in Python
  + devices securely communicate on a local network
  + especially important for embedded developers
* Database Local Caching
  + recently accessed data is cached in memory on a single node
    - cache “miss”
  + session state is maintained on an individual server
  + the cache is local to each app node
    - cache data is not shared with other local cache nodes
* Database Remote Caching
  + uses dedicated caching servers
    - the cache does not reside on the app node
    - this is what ElastiCache does
  + designed to support intensive concurrent data requests
  + it is a cluster of connected nodes
    - all nodes know what is cached, a “distributed cache”
  + integrated caching
    - cached data is kept consistent with data on disk
* Web Caching
  + goals –
    - reduce network traffic
    - reduced load placed on content origin servers
  + Content Delivery Network
    - CloudFront
    - data is replicated to geographical edge locations
    - user’s requests are satisfied from the edge location
  + cache origin servers
    - map or may not add HTTP header caching directives
  + client-side
    - web browser locally caches HTTP responses and content
    - reduces network traffic
  + server-side
    - reverse proxy caching
      * load on the origin server is reduced
      * hides identity of origin servers
* ElastiCache
  + fully-managed AWS caching solution
  + open source in-memory data store
  + improves data retrieval performance
* Redis for ElastiCache
  + supports **multiple** data types
  + BSD licensed
  + in-memory data structure store
  + built-in replication
  + on-disk persistence settings
* Memcached for ElastiCache
  + supports Strings as a data type
  + simple yet powerful in-memory key-value store for small chunks of arbitrary data
  + data comes from results of database calls, API calls, or web page rendering
  + its simple design allows for
    - quick deployment
    - easy web development
* Session Management and Caching
  + HTTP is inherently stateless
    - client connects to server, posts or gets data, then disconnects
  + HTTP session state can be maintained
    - client-side with web browser cookie
    - server-side with database
  + stick sessions or session affinity
    - user traffic is routed to the same back-end server
      * session state is restored here
      * if the server fails, session state is lost
    - horizontal scaling could result in unequal workload distribution
  + distributed session management
    - remove session management from individual servers
    - uses an in-memory cache
      * Redis or Memcached
    - multiple nodes handle session management
    - session data can be replicated to other nodes
* CloudFront
  + global web application content caching from origin servers
    - all content
    - filtered content
  + content is copied to various AWS regions
  + allows for quick loading of web site content
  + origin types
    - S3 bucket
    - web server
    - AWS MediaPackage channel endpoint
    - etc
  + cache expiration settings
    - cached date is not compared to origin data for every request unless the TTL is execeeded
    - default TTL is 1 day
      * used when origin does not supply HTTP header cahce directives
    - maximum TTL is 1 year
      * used when origin does supply HTTP header cache directives
* Hybrid Environment Caching
  + hybrid means linking on-premises and cloud resources
    - extends an o-p network to the cloud
  + connection types
    - Virtual Private Network VPN – encrypted tunnel over public internet
    - Direct Connect – private dedicated network circuit (more secure)
  + caching can speed up data access
    - o-p access to cloud data
    - cloud access to o-p data
  + o-p AWS Storage Gateway
    - commonly used an o-p VMware or Hyper-V virtual machine wit hstorage
    - policies determine which data is cached on the device
    - o-p apps use standard protocols to access data
      * AWS cloud awareness is not required for o-p apps

Cloud Scalability

* Horizontal Scaling
  + auto scaling
    - can also be used with a load balancer
  + In – reduce the number of EC2 instances
  + Out – increase the number of EC2 instances
* Vertical Scaling
  + Up – change instance type to increase power
  + Down – change instance type to decrease power
    - vCPUs and Disk IOPS
* High Performance Computing (HPC)
  + execute large complex compute jobs in parallel across a HPC cluster
  + Elastic HPC Clusters
    - cost savings through EC2 spot instances
    - shared storage for the cluster using Amazon EFS
    - can be deployed via CloudFormation templates
    - Web GUI management using EnginFrame
  + Scalability
    - the ability to handle the changing needs of IT services for an organization
    - example – increasing loads on a system
  + Elasticity
    - refers to increasing or decreasing cloud resources
    - may not be ideal for all environments
      * example – environments that do not experience sudden changes

AWS Lambda

* compute service
* executes code
  + only when needed
  + scales automatically
* costs – only when running
* any type of application/backend service
  + run code virtually
  + zero administration
* high-availability compute infrastructure
  + administration of compute resources
    - server/os maintenance
    - provisioning/scaling
    - monitoring/logging
* Lambda-based application
  + lambda function
    - core of the Lambda base application
    - triggered from an event source
  + event source –
    - something that has happened
    - ex file uploaded to amazon
  + downstream resources
    - actions are then taken on downstream resources before or during the execution of the Lambda function
  + log streams
    - generated to be used for monitoring and debugging
  + AWS SAM
    - defines a standard application model for serverless applications
* Tools
  + Lambda Console
  + AWS CLI
  + SAM Local
* Code deployed to Lambda as a –
  + JAR file
  + ZIP file
* Requirements
  + write application code (node, java, c#, or python)
  + run within runtime environment
  + responsible for your code
  + cannot log in to compute instances, or customize operating systems, or language runtime
* Lambda Function Concepts
  + Handler
    - starts execution of Lambda function
  + Context Object
    - code interacts with AWS Lambda
    - async platform
      * uses call backs
  + Logging
    - CloudWatch Logs
    - log entries
  + Properties
    - Lambda function:
      * communicate results
    - Stateless Style
      * compute infrastructure
    - Local File System Access
    - Persistent State
      * stored in Amazon S3
      * Amazon DynamoDB
      * other cloud storage service
* Serverless Web Hosting
  + runs on-demand
  + unlimited capacity
  + automatic scaling (pay for what you need)
  + runs on managed AWS infrastructure
  + code runs in up-to-date and secure Environment
  + great for SPA and API apps
  + limited support for Fullstack Apps
* API Gateway
  + no code, from management console
  + create resource (instead of path or endpoint)
    - create method
      * pick HTTP option
      * pick Integration type
        + lambda function
        + http
        + mock - dummy
        + aws service
      * integration response
        + can give back data
        + ex JSON message “this is working”
      * choose a stage

AWS CodeCommit

* private Git repository on AWS
* scalable and highly available
* works with existing Git tools
* uses IAM for access control
* integrates with AWS services

AWS CodeBuild

* fully-managed build service
* compile, test, and package
* operates from Console, CLI, and SDK
* can utilize custom build environment in Docker image
* integrates with AWS services

AWS CodeDeploy

* automates build deployments
* can deploy to on-premises environments
* scales with environment size
* stop and rollback features
* language and architecture agnostic
* integrates with AWS services
* Deployment Types
  + blue/green deployment
    - replaces the instance in the deployment group with new instances and deploys the latest application revision to them
  + in-place deployment
    - updates the instances in the deployment with the latest application revision. each instance will be briefly taken offline for its update

AWS CodePipeline

* CI/CD Service for release management
* workflow model for application updates
* allows parallel workflow branches (ex functional and stress test at same time)
* custom and conditional functions
* integrates with AWS services and many popular third-party services like Jenkins

CodeCommit DeepDive

* Git
  + open source version control system
  + VCSs keep files and list of changes
  + Git keeps snapshots, not differences
  + resources have 3 states
    - modified – file does not exist in repo or has been changed but not saved
    - staged – file to be included in snapshot
    - committed – file recorded in snapshot
  + clone, add/change, commit, push
  + git requires credentials
  + code commit services
    - development machine
    - AWS CLI CodeCommit Console
      * create repository
    - CodeCommit Service
      * create remote repository

Amazon Cognito

* Identity Provider and Identity Broker
* IDP Identity Provider service that you can use to add authentication and authorization to mobile and web applications
* Identity Broker that can generate temporary access credentials after user has been authenticated by 3rd party IDP
* Cognito is a serverless service
* development tool that offers JS, iOS, and Android SDK
* out-of-the-box integration API Gateway
* supports these flows
  + sign-up/sign-in with MFA (Multi-Factor Authentication)
  + profile management
  + forgot password
  + email and mobile verification
* customizable through lambda functions
* allows guest users
* uses Secure Remote Password protocol
* Workflow Customization – User Pool
  + Registration Lambda Triggers
    - pre-signup
    - registered
    - custom message
    - confirmation sent
    - confirmed
    - post-confirmation
  + Authentication Lambda Triggers
    - pre-auth
    - custom message
    - MFA/Challenge (challenge can be a secret question)
    - Authenticated
    - Post-auth
  + Custom Auth Lambda Triggers
    - define auth challenge
    - create auth challenge
    - verify auth challenge response
* Authentication Flow
  + User Pools – scalable user directory service that you can use to add authentication and authorization to mobile and web applications
  + Federated Identity – Identity Broker than can generate temporary access credentials after user has been authenticated by 3rd party IDP
  + User Pool – authentication flow
    - 1.1 Initiate Authentication
    - 1.2 Return JWT Token
      * could also return a challenge or error
    - 2.1 Use ID Token to access APIs
    - 3.1 Use Token to access AWS Resources
  + Federated Identity – *Classic* authentication flow
    - 1.1 Login
    - 2.1 GetID (pass OpenID1)
    - 2.2 Validate
    - 3.1 Assume Role (pass OpenID2)
    - 3.2 Validate
    - 4.1 Access AWS Services
  + Federated Identity – *Enhanced* authentication flow
    - 1.1 Login
    - 2.1 GetID
    - 2.2 Validate
    - 3.1 Get Credentials
    - 3.2 Validate
    - 4.1 Access AWS services
  + Tokens
    - after authentication client gets 3 tokens
    - **ID Token** used to authenticate with AWS services and backend applications
    - **Access Token** used to authorize operations on user’s account (profile management)
    - **Refresh Token** used to refresh ID or Access tokens with Cognito service
    - tokens follow JSON Web Key Token JWT format

AWS CI/CD

* automatic deployment
* possibility to go into different stages
* CodeCommit : storing our code
* CodePipeline : automating our pipeline from code to end
* CodeBuild : building and testing our code
* CodeDeploy: deploying the code to EC2 fleets
* a testing/build server checks the code as soon as its pushed
* the devs gets feedback about the tests and checks that passed/failed
* Continuous Integration
  + to find bugs early, fix bugs
  + deliver faster
  + deploy often
  + unblocked devs
* Continuous Deployment
  + ensure software can be reliably released
  + ensures deployments happen often and quickly
* Code (CodeCommit) – Build (CodeBuild)– Test (CodeBuild) – Deploy (CodeDeploy)– Provision
  + Orchestrate – AWS CodePipeline

AWS CodeCommit

* Version Control
  + the ability to understand the various changes that happened to the code over time (and possibly roll back)
  + enabled by using a version control system such as Git
  + can reside on machine or on a central online repository
  + ensures code is backed-up, fully viewable, and auditable
* a private Git repository
* no size limit (scale seamlessly)
* fully manageable, highly available
* code only in AWS Cloud account –> increased security and compliance
* secure – encrypted, access control, etc
* integrated with Jenkins/CodeBuild and other CI Tools
* **interactions are done using Git**
* Authentication in Git:
  + SSH Keys – users can configure SSH keys in their IAM Console
  + HTTPS – done through the AWS CLI Authentication helper or generating HTTPS credentials
  + MFA (multi factor authentication) – can be enabled for extra safety
* Authorization in Git:
  + IAM Policies manage user/role rights to repo
* Encryption
  + repos are automatically encrypted at rest using KMS
  + encrypted in transit (can only use HTTPS or SSH – both secure)
* Cross Account access:
  + do not share your SSH keys or AWS credentials
  + use IAM Role in your AWS Account and use AWS STS (with AssumeRole API)
* vs GitHub
  + Security
    - GitHub: GitHub Users
    - CodeCommit: AWS IAM users & roles
  + Hosted
    - Github: hosted by Github
      * Github Enterprise: self hosted on servers
    - CodeCommit: managed and hosted by AWS
  + UI
    - CodeCommit has minimal UI
* CodeCommit Notifications
  + you can trigger notifications in CC using AWS SNS (Simple Notification Service) or AWS Lambda or AWS CloudWatch Event Rules
  + use cases for SNS/ AWS Lambda notifications
    - deletion of branches
    - trigger for pushes that happens in master branch
      * ex ensure secret credentials weren’t committed
    - notify external Build System
    - trigger AWS Lambda function to perform codebase analysis
  + use cases for CloudWatch Event Rules
    - trigger for pull requests updates (CUD + commented)
    - commit comment events
    - CW E R goes into an SNS topic
* CC USE
  + “$” AWS 🡪 CodeCommit
  + “$” Create repository
    - commit message
    - upload code
  + Configure Triggers (notifications)
  + Add SSH Key
    - “$” Services (top left drop down)
      * “$” Users
      * Upload SSH public key
  + /OR/ HTTPS Git credentials
    - “$” Services (top left drop down)
      * “$” Users
      * Generate (HTTPS Git credentials for AWS CodeCommit)
  + Git Bash $ cp ../nodejs-vs-blue/\* . //copies all files from that folder, cp = copy
  + $ git add .
  + $ git commit –m “asdf”
  + $ git push

CodePipeline

* continuous deliver
* visual workflow
* Source: GitHub / CodeCommit / amazon s3
* Build: CodeBuild / Jenkins / etc
* Load Testing: 3rd party tools
* Deploy: AWS CodeDeploy/Beanstalk/CloudFormation/ECS
* made of stages
  + each stage can have sequential actions and/or parallel actions
  + stages example- Build/Test/Deploy/Load Test
  + manual approval can be defined at any stage
  + **stages have multiple action groups**
* AWS CodePipeline Artifacts
  + each pipeline stage can create ‘artifacts’
  + artifacts are passed and stored in Amazon S3 and passed on to the next stage
  + trigger 🡪 Source (CodeCommit) –OA🡪 Amazon S3 bucket –IA🡪Build(CodeBuild) –OA🡪 Amazon S3 bucket –IA🡪Deploy (CodeDeploy)
    - OA = output artifacts, IA = input artifacts
* CP Troubleshooting
  + CodePipeline state changes happen in AWS CloudWatch Events, which can in return create SNS notifcations
    - ex create events for failed pipelines
    - ex create events for cancelled stages
  + if CodePipeline fails a stage, your pipeline stops and you can get information in the console
  + AWS CloudTrail can be used to audit AWS API calls
  + if Pipeline cant perform an action, make sure the “IAM Service Role” attached has enough permissions (IAM Policy)
* CP USE
  + “$” Service role
  + Artifact store
  + Source Provider
  + Change detection options – default for all
  + “$” Edit (top of the page after creation, in middle [far left of options])
    - Add stage
    - Add action group
      * can pick Manual approval
    - Add 2nd action group
      * can add below so that it is sequential and will only happen after 1st action is completed
      * /OR/ add to the side so that the actions run in parallel
    - “$” Save (top right of page)
* CP Amazon
  + Define your release process models
    - describe release workflow
    - define the stages your code will progress through
    - define the actions performed during each stage
  + next time changes are made to your source repository
    - they’re pushed through that defined set of actions
    - they’re approved before progressing down your pipeline
  + Best Practices
    - create templates
    - take advantage of AWS integrations

CodeBuild

* fully managed build service
* continuous scaling (no servers to mange or provision – no build queue)
* pay for usage (the time it takes to complete the builds)
* leverages Docker under the hood for reproducible builds
* possibility to extend capabilities leveraging our own base Docker images
* Secure: integration with KMS for encryption of build artifacts, IAM for build permissions, and VPC for network security, CloudTrail for API calls logging
* source code form Github/ CodeCommit/ CodePipeline/ S3
* build instructions can be defined in code (buildspec.yml file)
* outputs log to Amazon S3 & AWS CloudWatch logs
* metrics to monitor CodeBuild statistics
* use CloudWatch Alarms to detect failed builds and trigger notifications
* CloudWatch Events/ AWS Lambda as a Glue
* SNS notifications
* ability to reproduce CodeBuild locally to troubleshoot in case of errors
* builds can be defined within CodePipeline or CodeBuild
* can use Docker to extend any environment
* how CodeBuild works
  + Source Code (CodeCommit) **buildspec.yml**
  + **Build Docker Image** (AWS Managed or Custom)
  + CodeBuild Container
    - running on Build Docker image
    - running instructions from buildspec.yml
  + AWS S3 Cache Bucket (optional to increase performance)
  + outputs to AWS S3 Bucket Artifacts
  + saves logs to AWS CloudWatch AWS S3
* CodeBuild BuildSpec
  + buildspec.yml file must be at the root of your code
  + define environment variables
    - plaintext variables
    - secure secrets: use SSM Parameter store
  + Phases (specify commands to run):
    - install – install dependencies you may need for your build
    - pre build – final commands to execute before build
    - build – actual build commands
    - post build – finishing touches (ex zip output)
  + Artifacts – what to upload to S3 (encrypted with KMS)
  + Cache – files to cache (usually dependencies) to S3 for future build speedup
  + CodeBuild Local Build
    - troubleshoot with CodeBuild Agent
* CB USE
  + “$” Getting Started
  + Project Configuration
  + Source – CodeCommit
  + Environment – managed or custom Docker
  + Operating System- Ubuntu?
  + Image/Image version – use latest
  + New service role
    - role name
  + VPC optional
  + Environment variables – optional
  + Buildspec
    - Use a buildspec file
  + Artifacts
    - Encryption key
  + “$” Start Build
  + Add yaml file to code repository (CodeCommit)
    - inline $
    - phases:
      * install:
        + runtime-versions:

nodejs: 10

* + - * + commands:

- echo “installing”

* + - * build:
        + commands:

grep –Fq “Congratulations” index.html //file must have congratulations inside or else build fails

* + shows all phases with logs for each
  + and CloudWatch logs
  + integrate by going back to CodePipeline
    - add stage
* CodeBuild in VPC
  + by default, CodeBuild containers are launched outside your VPC
  + therefore, it cannot access resources in a VPC (by default)
  + you can specify a VPC configuration:
    - VPC ID
    - Subnet IDs
    - Security Group IDs
  + then your build can access resources in your VPC (RDS, ElasiCache, EC2, etc)
  + uses cases: integration tests, data query, internal load balancers

AWS CodeDeploy

* automates code deployment
* each EC2 Machine (or On Premise machine) must be running the CodeDeploy Agent
* the agent is continuously polling AWS CodeDeploy for work to do
* CodeDeploy sends appspec.yml file
  + appspec.yml must be at root of source code
  + specifies instructions on how to deploy
* application is pulled form GitHub or S3
* EC2 will run the deployment instructions
* CodeDeploy Agent will report of success/failure of deployment on the instance
* Steps
  + 1. push to GitHub/Amazon S3
  + 2. trigger deployment to AWS CodeDeploy
  + 3. EC2 instances + agent poll to CodeDeploy
  + 4. S3/GH downloads code + appspec.yml file from EC2 instances
* EC2 instances are grouped by deployment group (dev/test/prod)
* lots of flexibility to define any kind of deployments
* CodeDeploy can be chained into CodePipeline and use artifacts from there
* CodeDeploy can re-use existing setup tools, works with any application, auto scaling integration
* Blue/Green only works with EC2 instances
* support for AWS Lambda deployments
* CD does not provision resources
* CodeDeploy Primary Components
  + application: unique name
  + compute platform: EC2/On-Premise or Lambda
  + deployment configuration: deployment rules for success/failures
    - EC2/On-Premise: can specify the minimum number of healthy instances for the deployment
    - AWS Lambda: specify how traffic is routed to your updated Lambda function versions
  + deployment group: group of tagged instances (allows gradual deployment)
  + deployment type: in-place deployment or blue/green deployment
  + IAM instance profile: need to give EC2 the permissions to pull from S3/ Github
  + Application Revision: application code + appspec.yml file
  + Service role: role for CodeDeploy to perform what it needs
  + Target revision: target deployment application version
* CodeDeploy AppSpec
  + File Section: how to source and copy from S3/GitHub to file system
  + Hooks: set of instructions to do to deploy the new version (hooks can have tiemouts). the order is –
    - ApplicationStop
    - DownloadBundle
    - BeforeInstall
    - AfterInstall
    - ApplicationStart
    - **ValidateService**- health check
* Deployment Configuration
  + Configs
    - one at a time- 1 instance at a time, if an instance fails, deployment stops
    - half at a time
    - all at once- quick but no healthy host, downtime. good for dev
    - custom- min healthy host = 75%
  + Failures
    - instances stay in “failed state”
    - new deployments will first be deployed to “failed state” instances
    - to rollback- redeploy old deployment or enable automated rollback for failures
  + Deployment Targets
    - set of EC2 instances with tags
    - directly to an ASG
    - mix of ASG/Tags so you can build deployment segments
    - customization in scripts with DEPLOYMENT\_GROUP\_NAME environment variables
* CodeDeploy Use
  + “$” select your use case
    - CodeDeploy
    - CodeDeploy – ECS
    - CodeDeploy for Lambda
  + Create 2 roles
    - first role auto attaches, AWSCodeDeployRole
      * role that allows CodeDeploy to call AWS services
    - another role for EC2 service
      * must be able to pull data from S3 instance
      * can give read-only access – AmazonS3ReadOnlyAccess
  + “$” Create application (orange button on right under getting started)
    - choose platform
      * choose EC2 since that was the role we created (probably lambda for us)
      * add tags
      * configure Security Group
  + install CodeDeploy Agent
    - $ sudo yum update
    - $ sudo yum install ruby
    - $ wget <https://aws-codedeploy-eu-west-s3.s3.eu-west-3.amazonaws.com/latest/install>
      * can change region
    - $ chmod +x ./install //makes it executable
    - $ sudo ./install auto //executes
    - $ sudo service codedeploy-agent status //checks to see if CodeDeploy Agent started
  + Add tags
  + “$” Create deployment group (orange button on right)
    - add role (1st role created- CodeDeployServiceRole)
    - In-place (updates instances) or blue/green (replaces)
    - “$” Environment configuration
      * Amazon EC2 instances (for this)
    - Deployment Configuration
      * AllAtOnce // or another
    - Create bucket (for code)
      * the code must contain appspec.yml at root
      * the code contains hooks
    - Upload zip(or jar) file to S3 bucket
  + Rollbacks
    - can specify automatic rollback options
      * when a deployment fails
      * alarm thresholds are not met
      * or disable rollbacks
    - if a roll back happens, CodeDeploy redeploys the last known good revision as a new deployment
* CI/CD walkthrough
* Create New Pipeline
  + can use existing service role or it will create new one
  + add source provider- CodeCommit
  + repo name
* CodeBuild
  + source – CodeCommit
  + repo – same repo as before (Lambda)
    - branch- master
  + runtime- Standard
  + Role ARN – add permissions
  + build specifications – buildspec.yml
    - java: corretto11
  + Artifacts-
    - save Jar file
    - and buildspec.yml
    - save to Amazon S3 (or other)
* Deploy
  + deploy to different S3 file
  + pull from lambda deploy bucket
  + + extract file before deploy
  + provider - CodeBuild
* Edit Pipeline
  + add stage
    - deployment
* Build- 2nd file
  + source – CodeCommit
  + operating system – Ubuntu (could be other)
  + existing service role
  + no artifacts or bucket because they are already saved
* Pipeline
  + add action group
    - provider- CodeBuild
    - Input Artifacts – none against since they are already saved

**buildspec.yml**

version: 0.2

phases:

install:

runtime-versions:

java: corretto11

pre\_build:

commands:

- echo Nothing to do in the pre\_build phase

build:

commands:

- echo Build started on ‘date’

post\_build:

commands:

- echo Build completed on ‘date’

- aws lambda update-function-code

--function-name LambdaFunctionName

--ss3-bucket bucketName

--s3-key NewLambdaBuild/target/test.jar

artifacts:

files: