Certified DevOps Professional – Notes

**Introduction:**

* $300 and has a 30-day retake policy
* Can only take it 3 times in 12 months
* 170 Minutes long (2.1 minute per question)
* 80 questions
* Questions are long
* Appropriate level of information in the key exam areas
* Will only cover up to 70% of exam. Practical expertise and general knowledge. DevOps Knowledge
* Re-watch all videos multiple times. Read, YouTube videos, white papers, practice!
* **GIT, Software development concepts are other courses that may help**
* **May be good to do the SA pro course along side this one.**
* 8-10 hours of practical reading based on 1 hour of lectures.
* Other Courses that will help:
* https://www.youtube.com/user/AmazonWebServices/playlists
* DEVOPS - https://www.youtube.com/playlist?list=PLhr1KZpdzukeH9VMPbNHMCXl\_NrVc1JGe
* Develop Tools - https://www.youtube.com/playlist?list=PLhr1KZpdzuke5pqzTvI2ZxwP8-NwLACuU
* Databases - https://www.youtube.com/playlist?list=PLhr1KZpdzukeMbjRqGswHX38DCqOHZ5GA
* Compute - https://www.youtube.com/playlist?list=PLhr1KZpdzukfVW6NrpDzdT6Sej0p5POkN
* http://cantrill.io
* http://ozaws.com
* https://read.acloud.guru
* https://serverlesscode.com/
* https://paulwakeford.info/
* https://aws.amazon.com/blogs/aws/
  + https://www.awsarchitectureblog.com
  + http://blogs.aws.amazon.com/application-management
  + http://blogs.aws.amazon.com/security/
  + https://aws.amazon.com/blogs/compute/
* <https://aws.amazon.com/documentation/>
  + EC2 – instance roles, defaults, performance limitations
  + EBS – performance, limits, snapshots
  + S3
  + Cloudwatch – 20% of exam
  + CloudFormation – wait condition handlers, hold condition handlers
  + OpsWorks
  + Elastic Beanstalk
  + DynamoDB – CLI/UI perspective, partitioning
  + CloudTrail – auditing, data output locations
  + IAM
  + SQS, DataPipeline, Cognito, SNS
* <https://aws.amazon.com/whitepapers/>
* Practical Tasks
  + Cloud formation to deploy a HA wordpress instance
  + Cloud formation to deploy a php website, inside an auto-scaling group, reading from dynamo DB, then deploy a HTTP load-testing application, watch and manipulate the autoscaling
  + Write small lambda function, use it as a backing for customer resource in a cloud formation template.
  + Cloud formation template, update, replace, interrupt
  + Download EB example application, make changes, create DEV and PROD EB environments, make changes, and observe updates
  + Deploy 2 instances with appropriate roles, bootstrap the cloud watch logs agent and configure detailed log ingestion into cloudwatch.

**Core Concepts:**

* AWS CLI/API
* SLDC (Software Development Lifecycle)
* Continuous Integration, Build, delivery and deployment
  + Continuous Integration (CI) & Continuous Deployment (CD)
  + Problems with GIT occurred by changes made by other developers being incompatible and causing compile failures. Known as integration hell.
  + Longer the code was checked, the greater issue could arise with more compile issues
  + **Continuous Integration** – process of automating regular code commits followed by an automated build and test process designed to highlight integration issues early.
    - Requires additional tools like Bamboo, Cruise Control, Jenkins, Go and Team City
    - Customizable Workflow based integration
    - Spot prices could be a good fit for compilation and testing servers.
  + **Continuous Deployment** – takes the form of a workflow based process which accepts a tested software build payload from a CI server.
    - CD Server automates the deployment into a working WA, Pre-Production or production environment.
    - CodeDeploy and CodePipeline provide CI/CD services. Same with Elastic Beanstalk and Cloudformation.
  + Developers -> Source -> Build -> Staging -> Production -> Customers – Ideas Requests Bugs -> Changes Update Fixes
* Deployment Types
  + **Single Target Deployment**
    - Not used great these days
    - Mainly for Small development projects, legacy or non-highly available HA infrastructure
    - Build -> Target
    - Brief outage when version installed. Testing limited. Rollback involves removing the new version and installing the previous one.
  + **All-at-once deployment**
    - Happens in one step but destination is multiple targets
    - More complicated than single target, requiring orchestration tooling.
    - No ability to test, more for small deployments. Small outages and less than ideal rollback.
  + **Minimum in-service style deployment**
    - Happens in multiple stages
    - Deployment occurs to as many targets as possible while maintain minimum targets
    - Moving parts with orchestration and health checks required
    - No downtime
    - Quicker and less stages
    - Allows automated testing, targets assessed and tested prior
  + **Rolling deployment**
    - Happens in multiple stages but user defines number of targets
    - Moving parts with orchestration and health checks required
    - Applicable health isn’t necessarily maintained.
    - Can be least efficient deployment based on time-taken
    - Allows automated testing, targets assessed and tested prior
    - No downtime
    - Can be paused allowing limited multi-version testing.
  + **Blue/Green deployment**
    - Requires advanced orchestration tooling
    - Extra cost
    - Rapid deployment process.
    - Cutover/migration can be clean (DNS Change)
    - Rollback (DNS regression)
    - Health and performance of entire green field can be tested
    - Can be fully automated via template systems.
    - Binary
  + **Know Pro/Cons for Exam**
  + **Know when each should be used and when not**
  + **Know the limitations of each, how quick deployment, how quick rollback**
  + **Know how each deployment type impacts your applications**
  + **Known which AWS service support deployment type.**
* A/B Testing
  + **Sends a percentage of traffic to green/blue environment**
  + Separates different versions of your code.
  + Can allow testing/feedback to come from users
  + Allows gradual performance/stability/health analysis
  + New features can be tested.
  + Uses Route53 with 2 records in simple mode. Later switched to weighted.
    - DNS, caching, other DNS related issues can impact overall accuracy of technique.
* Bootstrapping
  + **Bootstrapping** – process during which you start with a base image, ISO/AMI, and automation build on it to create a more complex object.
  + CFINIT or CLOUDINIT
  + AMI based approach would require a lot of AMI’s.
  + Bootstrap can be done via cloud formation.
  + Help to bring all the components together like a cake.
  + Quick launch versions AMI’s use pre-built AMI’s with minimal configuration changes.
* Immutable Architecture
  + Immutable Architecture – practice of replacing infrastructure instead of upgrading or repairing faulty components.
  + Treat servers as unchangeable objects
  + If something develops a problem, diagnose, fix and return to service.
  + Treats servers as throwaway objects. If a failure happens, remove the server and create a new one from an AMI.
  + Never work manually.
  + Traditional architecture is like pets. Immutable servers are like cattle.
* Containers & Docker Primer
  + **Virtualization**
    - Guest OS, Dependencies, Application, VM
    - Wasted space
  + **Containerization**
    - Dependencies, Appliance, Container.
    - Higher density and improved portability
    - Escape from dependencies.
    - Consistent progression from Dev->Test->QA-Prod
    - **Isolation** – performance or stability issues with App A in Container A, won’t impact App B in Container B
    - Resource scheduling at the micro level.
    - Code portability
    - Micro-Services
  + **Docker Components**
    - **Docker Image** – basis of a docker container ISO. Read only.Base Build docker containers.
    - **Docker Container** – holds everything needed to make an application to work.
    - **Layers/Union File System** – Combines layer into a single image. Branches are separate file systems.
    - **Docker File** – instructions create or include each layer. Stored in a docker file.
    - **Docker Daemon/Engine** – create OS to run your applications. Communicates with the docker client to build/ship/run containers
    - **Docker Client** – interface between you and the engine. Control docker daemon
    - **Docker Registries /Docker Hub** – hold images in a repo. Provided by Docker Hub. Can use images based on what others have done.
* JSON Primer
  + **JSON (JavaScript Object Notation)** – way to represent structured data for interchange between appliances.
  + Used most often with Web services like Rest API
  + **Name/Value pairs** – consists of key followed by a value
  + Can be a string, array, object, null value, JSON structure
  + **Object** – collection of key/value pairs.
  + **Array** – ordered list of values surrounded by values.
  + **JSON String** – contains an array of values or an object
  + **Policy document (JSON string)** – complicated information contained by nested objects.

**CI/CD/Automation:**

CloudFormation Primer

* **CloudFormation** – building block service designed to provision infrastructure (cfn)
  + ElasticBeanstalk using cloudformation
  + JSON based. **Know how to read/write JSON.**
  + Let cfn name the resources used in the template.
  + Can self cleanup
* **Stack** –cloud formations unit of grouping for infrastructure. Controls lifecycle of the infrastructure.
  + Has stack ID. Can be applied many times.
* **Template** – JSON document giving cloud formation instructions on how to act and what to create. Used to update or create.
  + Template limit of 200.
* **Stack Policy** – IAM style policy which governs what can be changed and by who. (cfnStack)
* **Parameters –** allow the passing of variables into a template via UI, CLI, or API.
  + Can have a number of attributes like Keypair, string, number, AZ
  + Can have a default value
  + Allowed values – one of more values which the parameter can take
  + Allowed pattern – regular expression that defines the format the parameter
  + Min & Max Value for numbers
  + Min & MaxLength for strings
  + **Look at documentation – UserGuide/parameters-section-structure.html**
  + Cloudformation can pick values if they are not specified in parameters
* **Mappings –** allow processing of hash’s (arrays of key value pairs) by the cfnTemplate.
  + Use case – define lookup to select ami id based on region.
* **Resources –** where your actual resources are declared.
  + Required in the template.
* **Outputs –** results from the template.
  + Can run scripts within the instances
  + Can have conditional elements to resources or whole resources conditional.
* Can expand files within instances.
* Always challenge yourself with infrastructure configuration.

Structure

* **Ref** – can reference an object within the template.
* Stacks – can have output values, parameter references or output function
* Get att can reference alternative values.
* **Work on Cloudformation**

Intrinsic Functions & Conditionals

* **Intrinsic functions** – inbuilt functions provided by AWS to help you manage, reference, and conditionally act upon resources, situations and inputs to a stack. Looking for max portability with Cloudformation templates.
  + Fn::Base64 – accepts plain text and converts to Base64 for EC2
    - { “Fn::Base64” : “yum –y update && yum “}
  + Fn::FindInMap – maps objects to one or more keys. Lookup function
  + Fn:GetAtt – looks at the non default values
  + Fn::GetAZs
  + Fn::Join
    - “Fn::Join”:[“:”,[“a”,”b”,”c”]]
  + Fn:Select – select an object from a list of objects
  + Ref – default value for resource
  + Avoid 2+ AZ if you want your template to work everywhere.
* Conditional Funtions
  + Fn::And – returns true if all input are true
  + Fn::Equals
  + Fn::If
  + Fn::Not – returns false if condition evaluates to true.
  + Fn::Or – return true if any inpute conditions are true

"VPC":{

"Type":"AWS::EC2:VPC",

"Properties":{

"CidrBlock":{"Ref":"VPCIPRange"}

}

}

"SubnetDMZA":{

"Type":"AWS::EC2::Subnet",

"Properties":{

"VPCid":{"Ref":"VPC"},

"CidrBlock":{"Ref":"IPRange1"}

"AvailablityZone":{"Fn::Select":{"0",{"Fn::GetAZs":""}}},

}

}

Stack Creation & DependsOn

* Template Upload/S3 Template Reference
* Template syntax check
* Stack name & parameter verification & ingestion
* Cloud formation template processing & stack creation
  + Resource ordering
  + Resource creation
  + Output Generation
  + Stack Completion or Rollback
* **DependsOn** – influence the automatic dependency checking of cloudformation
  + Allows you to direct cloud formation on how to handle more complex dependencies.
  + Uses this to allow remove/delete/rollback
  + References another resource but doesn’t use the reference function

CloudFormation Resource Deletion Policies

* Policy/setting which is associated with each resource in a template
* A way to control what happens to each resource when a stack is deleted
  + **Delete**
  + **Retain –** keep resources after deletion
  + **Snapshot –** restricted policy type and present in EC2, RDS, and Redshift. Takes a snapshot prior to deletion. Used with data processing workloads where critical elements are generated data. QA setup or QA run.
  + If not specified, the default is delete.
  + Defined at the top level of the resource.
* Transitive environment – can be instantiated and removed without change to your wider environment.
* Used in testing, CI/CD/QA workflows, presales, short life cycle/ immutable environments.
* Less billing control and resources are still charged after stack deletion.

Stack Updates

* Rights are checked and then updated. Stack policy controls.
* By default, absence of stack policy allows all updates
* Stack policies can’t be deleted once applied
* **Once a policy is applied, by default ALL objects are protected, Update:\* is denied**.
* **To remove the default DENY protection of an applied stack policy you need to update the policy with a explicit “allow” on one or more resources.**
* Can use NotResource for inverted logic.
* Principal where stack policies is required to be a wildcard
* Action – Update:Modify (no interruptions or some interruptions), Update:Replace (updates which cause resource replacement), Update:Delete, or Update:\*
* Update can impact a resource
  + No interruption – no impact to service
  + Some interruption – restarted or connectivity updated.
  + Replacement – changes are huge. Replaced with new object
  + Delete – resource removed from template.
* Cloudformation has the same limitation the infrastructure does in the template when being built out.
* Effect as with IAM policies like allow/deny
* Resource designated as single or wild card

Nesting

* **Nesting** – resource can be a whole stack nested in a parent template. Can have nested stacks.
  + Allows huge set of infrastructure split over multiple templates.
  + **460k template limit**.
  + **200 resources in 1 stack**
  + **100 mappings, 60 parameters, 60 output limit per stack.**
  + Allows more infrastructure as code reuse.
  + Sharepoint master template
    - SQL
    - AD & Infrastructure
    - Sharepoint 2013.
  + Steps
    - AWS::CloudFormation:Stack.
    - Template URL
    - Parameters
    - If there isn’t a default, and parameters are not defined, stack will fail.

CloudFormation creation policies, wait conditions and handlers

* **DependsOn** – used for controlling resource creation order within cloudformation.
  + Ready in console doesn’t mean functionality ready.
  + Fawled. Waits until dependencies continue.
  + **Creation Policies, Wait Conditions and Wait Condition Handlers** – influence WHEN a resource is marked as completed – delaying until its actually ready.
    - Creation policies can only be used on **EC2 and Autoscaling Groups**
    - 1st creation policy definition – important are DesiredCapacity and Count.
    - 2nd signal configuration of EC2 instance.
      * # of signal is => count in creation. It is marked as complete.
    - **Wait conditions** – resource that links the handler to the resource. 1. DependON key, 2. Handle property reference handle. 3. Response timeout. 4. Count.
      * Resources can depend on this
    - **Wait handlers** – cloudformation resource with no properties but it generates a signed URL
      * Additional data can be passed back to the signed URL.
    - Count – number of times a resource is reached when the wait handler is triggered.
    - Timeout – when the command timeouts.

Cloud Formation Custom Resource

* **Custom resource** – create any type of a AWS resource along with properties.
  + Not all AWS services are supported
  + It’s not just code. Doesn’t work with non AWS resources.
  + Ability to interact with external services.
  + Custom resources can help overcome the above listed.
    - Custom:ResourceNameHere
    - ServiceToken
  + Stack is created,updated, or deleted a SNS is set to a SNS topic with the event.
  + Cloudformation can call lambda functions in a certain region.
    - Cloudformation->SNStopic->Lambda or EC2 working or external application.
    - Stackeid.
    - ResponseURL
    - Request Type
    - Resource Type
    - Resource Prperties -> CIDR.
    - Status
    - Request ID
    - Physical and logical resource id
    - Data
  + Stack lined to on-premise resource creation
  + Stack linked to advanced logic – resource discovery
  + Stack deletion linked to advanced tidy operations – backups/monitoring deactivation
  + Stack linked to on-premise configuration management system.
  + Web stack creation – linked to external monitoring/penetration testing system.
  + Stack creation/deletion updates a lambda based backup solution – EBS snapshotting
  + Stack deletion spawns account wide pruning for orphaned EBS volumes.

OpsWorks Primer

* **OpsWorks** - AWS implementation of CHEF configuration management & Automation system
* Abstracts some of the detail when provisioning infrastructure
* Chef uses a config file that updates code across all servers
* More power and customization than Beanstalk. Uses JSON. More convenient
* **Stack** – collection of resources to perform a certain function.
* **Layer** – shared functionality and architecture which is shared to a group of components.
* **Instance** – actual units of compute. Inherit configuration from both stack and layer
* **Application** – deployed on 1 or more instances.
* **OpsWorks Agent (CHEF)** – responsible for configuration of machines
* **OpsWorks Automation Engine** – creation and deletion of various AWS infrastructure componenets
  + Auto scaling
  + Auto healing
  + Handle load blaancer
  + LifeCycle Events
* Sits inbetween cloudformation and Elastic Beanstalk.
* OpsWorks and CHEF are **declarative** desired state engines
  + Stat WHAT you want to happen and leave CHEF/OPSworks to handle the HOW
  + Recipes tell OPSworks WHAT you want the end result to be.
  + Cookbooks contain recipes and all associated data to support them.

OpsWorks Stacks and Layers

* **Stacks** – defined when creating one.
  + Name – names the stack
  + Region – choose the region and can’t be changed afterwards
  + VPC – controls the VPC in which OpsWorks instances are deployed
    - Need internet access to function. Otherwise they will fail.
    - Can’t be changed afterwards
  + Subnet – generates from the VPC
  + 12 stack allows you to choose linux or windows. Can’t mix OS vendors. OS can be changed later but won’t update existing instances.
  + Default SSH key – Other access keys
  + Specific a Git repo.
  + Advanced Settings:
    - Default root device
    - Host name theme is a vanity setting.
    - Agent version – latest is deployed
  + Resources:
    - Elastic IP’s
    - Volumes or RDS
* **Layers** – logical grouping of instances
  + General Settings
    - Layers Name, short name
    - Shut down time
    - Auto healing – enabled from the layer not the stack
    - Custom JSON – used in relation to chef databags
  + Recipes
  + Network
    - Allocate LB
    - Set private/public IP
    - Add an ELB when the EC2 instance. Can’t create one from the Opsworks console
  + EBS Volumes
  + Security
    - EC2 profiles
  + 3 different types of layers
    - **ECS** – a layer which allows integration of an ECS (Docker) cluster within OpsWorks
    - **RDS** – allows integration between OpsWorks an existing RDS single instance or HA pair.
      * **Can only be associate with one Opsworks stack**
      * **Stack clone operation doesn’t copy an existing RDS instance.**

OpsWorks LifeCyle Events

**Events** – can be executed by stack run command

* Each layer has its own recipes for each event.
* **Setup** – occurs when an instance has finished booting
* **Configure** – run on all instances when it enters or leaves online state or when EIP are associate or detach from LB.
* **Deploy** – deploys application to a set of server by
* **Undeploy** – delete an application or run an undeploy
* **Shutdown** – runs when an instance shutdown, but before it is terminated. Allows cleanup.

OpsWorks Instances

* Layer contains default recipes, general config, network config, and disk additions
* Instances can be added in two locations, the layer, or the stack instances menu.
* **24/7** - provisioned manually, and manually started or stopped by an admin. (can be controlled via CLI)
  + Configuring:
    - Subnet
    - SSH key
    - OS
    - Root Device Type
    - Volume Type
* **Time-based instances** – initially provisioned, and configured to power-on and off at certain times during the day.
  + Same as configuring 24/7
  + All times in UTC
  + Power-on at a specific time during a day will be light green.
  + Solid green will show times all day.
  + Start instances will turn green.
  + Current time will show an inverse triangle
* **Load-based instances** – initially provisioned and configured to automatically power on or off, based on configurable criteria.
  + Enable scaling **per-layer** scaling config.
  + Simple scaling based on CPU/MEMORY/LOAD
  + Complex scaling based on cloudwatch alarms

OpsWorks Applications

* **Application** – object which represents metadata and application. Below shows what can be choosen when setting it up.
  + Application Name
  + Documents Root
  + Data Source
  + Application Source
    - GIT
    - HTTP
    - S3
  + Environment Variables
  + Domain Names
  + SSL Enable & Settings
  + Deploying an App
    - Executes the deploy recipes on the instances targeted by the command
    - Passed to the common is the application-id
    - Application Parameters are passed into the chef environment within Databags
    - The deploy recipe accesses the application source information and pulls the application payload onto the instance.
    - 5 versions are maintained current and four historic.
    - **Create rollback** command can be used to roll back.

**Create-Deployment Command** – not limited to deployments

* Two main functions
  + Application deployments
  + Allows stack level commands to be executed against the stack.
* Syntax
  + **Aws opsworks –region us-east-1 create-deployment**
  + **–stack-id** – operation on a stack or reference towards it
  + **–app-id** – references an app and optional
  + **–instance-ids** – list of instances where the app will run
  + **–comment** – user defined comment string
  + **–custom-json** – operator to provide custom data and callbacks.
  + **–cli-input-json**
  + **–generate-cli-skeleton** – skeleton json structure and can be updated in json inputted. Automation
  + **–command** – execute for create deployment command and options for the option
    - install\_dependencies
    - update\_dependencies
    - update\_custom\_cookbooks
    - execute\_recipies
    - configure
    - setup
    - deploy
    - rollback
    - start
    - stop
    - restart
    - undeploy
* Deployments
  + Aws opsworks –region us-east-1 create-deployment –stack-id 1111-111-111-11 –app-id 111-111-111 –command “{\”Name\”:\”deploy\”}”
  + System maintains 5 version of the application. Current version and 4 historic versions
  + –command “{\”Name\”:\”undeploy\”}” – runs on all instances by the operation
  + –command “{\”Name\”:\”rollback\”}” – replace current version of the application with an older version
* Stack commands
  + **Update\_custom\_cookbooks** – cause instances to perform a full redownload. Update recipes
  + **Execute\_recipes** – does what it suggests
  + **Setup** – fault finding/diagnostics
  + **Configure** – run instance setup and recipes. Server discovery
  + **Update\_dependencies** – available in linux only. Not available in chef 12. Command forces installation of minor changes
  + **Upgrade\_operating\_system** – only available on linux.
  + **Read Opsworks FAQ**

OpsWorks Databags & Berkshelf

* **Berkshelf** – system that addresses chef’s shortcomings
  + Opsworks stacks which operate on older versions < 11.10 could only specify one custom cookbook. Limited ability to re-use community recipes. Either had to combine them in a repo or limit to a smaller subset of recipes
  + Chef 11.10 added BerkShelf, allowing you to install cookbooks from multiple repos.
  + How to implement?
    - Need to enable custom cookbooks in the stack level
    - Create berksfile in the repo
      * Yourcustomrepo/Berksfile
      * Source <https://supermarket.chef.io>
      * Cookbook ‘apt’
      * Cookbook ‘bleh;, git: ‘git://somewhere/bleh.git’
      * Cookbook ‘cookbook\_name’, [‘>=cookbook\_version’], [cookbookc\_options]
* **Databags** – global variable within chef or opsworks infrastructure. Contextual information within recipes
  + Global accessible JSON objects with multiple ones like STACK, LAYER, APP, INSTANCE
  + Data accessed via Chef **data\_bag\_item & search** methods within compute assets
  + Constructed by Custom\_JSON field from Opsworks
  + Can contain Strings, Booleans, Number, Lists, and JSON objects
  + Search method allows access via a search index
    - Aws\_opsworks\_app – App Databag
    - Aws\_opsworks\_layer – Layer Databag
    - Aws\_opsworks\_instance – Instance Databag
    - Aws\_opsworks\_user – Users databag, a set of users for a stack
  + Example
    - App = search(:aws\_opsworks\_app).first
    - App\_path = “/srv/${app{‘shortname’])”
    - Package “git” do
    - Node {“platform\_version”] == “14.04”
    - End
    - Git app\_path do
    - Repository app[“app\_source”][“url”]
    - Revision app[“app\_source”][“revision”]
    - End

OpsWorks Auto-Healing

* Opwsworks instance has an agent on it.
* Instances perform an ongoing heartbeat, if it fails, opsworks will treat the instance as unhealthy and perform autoheal
* EBS Backed – instance stopped – online->stopping->stopped->requested->pending->booting->online
* Instance store – can’t be stopped/started – online->shutting down->requested->pending->booting->online
  + Instance terminates
  + Root volume deleted
  + Launch new instance
  + Reattach EBS volumes
  + Assign new public and private ip
  + Reassociate any elastic IP
* Doesn’t do:
  + Won’t recover serious instance corruption. Damaged instances start with a “start\_failed” error.
  + Start/failed status requires manual intervention
  + Won’t update the OS
  + Not a performance response, failure response.

Elastic Beanstalk Primer

* **Beanstalk** – can deploy, monitor, and scale and application quickly
  + Provides developers or end users with ability to provision application infrastructure
  + Highly abstract focus towards infrastructure, focusing on components and performance – not configuration and specifications
  + Attempts to remove or significant simplify infrastructure management, allowing applications to deployed into infrastructure environments easily.
  + Uses cloudformation as it’s basis
  + Offers Basic or Enhanced health reporting similar to cloudwatch
    - Switching to enhanced causes a refresh of the entire environment
  + Different tiers: Software Configuration, Updates and Deployments, health, Networking, and Data Tier
* Key components:
  + Applications:
    - Applications are the high-level structure in beanstalk
    - Entire application is one EB application
    - Each logical component in your application can be a EB application or EB environment within an application.
    - Redeploys application against all instances in the group defined.
    - **Can change type and sizes of instances**
  + Environments:
    - Applications can have multiple environments (Prod, Staging, Dev, V1, V2, or functional type (front-end, back-end)
    - Environments can be either single instance or scalable
    - Environments are either **web server environments** or **worker environments.**
    - Always go with load balancing and auto scaling.
    - **Know options within EB to select.**
    - **Know application and deployment options**
    - **Know container options**
    - **Don’t create an RDS database in the data tier as it ties it to lifecycle events.**
    - **1 DB = 1 environment in EB**
    - Cloning does not clone an instance.
    - Supported
      * Node.JS
      * PHP
      * Python
      * IIS
      * Go
      * Tomcat
      * Ruby
      * Docker – can be used to use not supported languages
      * Java
  + Versions
    - Application version are unique packages which represent versions of apps
    - An application is uploaded to EB as an application bundle .zip
    - Each application can have many versions 1:M relationship
    - Application versions can be deployed to environments within an application.
      * URL’s are swapped

Extending Beanstalk using .ebextensions

* **.ebextensions** is a configuration folder within a Beanstalk application source bundle (.zip)
  + allows granular configuration of the EB environment and customization of the resources it contains
  + files within .ebextensions are YAML formatted and end with .config.
    - processed in ABC order
    - had a source and command section
      * command contains two objects – 91-setupcron and 03-change perm
      * 01-setupcron constructs a cron entry to add custom monitoring
      * 02-changeperm changes the file to executable to allow cron.d to run
      * Can use Ref and Fn::GetAtt referring to APP\_CONFIGkk
  + option\_settings – allow you to declare global configuration options
  + resource – allow you to specify additional resources to provision in your environment or define granular configuration on those resource
  + packages, sources, files, users, groups, commands, container\_commands and service allow customization of the EC2 instance as part of your environment like AWS::CloudFormation::Init
* Leader instance
  + An EC2 instance within a Load-balancing Autoscaling environment like leader/master
  + Only during the environment creation
  + All nodes are equal after establishment
  + **Leader\_only** directive can be used on within the container commands section of .config. Runs only once.

Docker in Elastic Beanstalk

* Application Source Bundle
  + Application Source
  + Dependancies
  + Scripts
  + .ebextensions
* **Dockerfile** – defines structure of the docker container for custom container
  + Defines source image
  + Defines port which the **container** listens via the EXPOSE directive.
  + **Dockerrun.aws.json** – defines how to deploy an existing docker registry stored container as an EB application
    - Contains .dockercfg file for authentication
    - Container mapping with EC2
    - Shows EB how to user the container
  + **.dockercfg**
    - stored on S3. Must be in the same region.
    - use docker login registry-url to generate the config.json
    - private registry

**Monitoring/Metrics/Logging:**

Intro

* Cloudwatch Basics
* CloudWatch Custom Metrics
* CloudWatch Logs
* CloudWatch Log Filters
* Cloudwatch Alarms

Cloudwatch Basics

* Metric gathering service
* Monitoring/alerting service
* Graphing service
* Cloudwatch will remember metrics **for 2 weeks.**
* **Namespaces** – containers for metrics. Additional namespaces can be enabled by detailed monitoring.
  + You can look at metrics across your autoscaling group. Can aggregate by it as well.
  + **Read entire CloudWatch Developer Guide**

CloudWatch Custom Metrics

* Demo of deploying an EC2 ubuntu machine
  + Install python-pip and git
  + Pip install awscli
  + Git clone https://github.com/ACloudGuru /resources
  + date --utc “+%FT%T.%N” | sed –r ‘s/{{:digit:]]{6}$/Z/’

CloudWatch Alarms

* CloudWatch
  + Initiate actions on your behalf based on parameters you specify against metrics you have in use.
  + Actions sent to SNS or Autoscaling
  + Alarm period should be equal or greater than the metric frequency. Alarms can’t invoke actions because they are in a state, the state must change.
  + Alarms actions must be in the same region as the alarms
  + AWS resources don’t send metric data to Cloudwatch under certain conditions
  + States:
    - **OK** – Metric matches threshold defined
    - **Alarm** – metric is outside threshold data defined
    - **Insufficient\_Data** – Metric isn’t available or not enough data to determine alarm state
  + Can have up to 5000 alarms per AWS account
  + Can create or update an alarm via **mon-put-metric-alarm**
  + Can enable and disable alarms via **mon-[enable|disable\-alarm**
  + Can describe alarms via **mon-describe-alarms**
  + **Head -11 resources/alarms-commands.txt** shows commands for Cloudwatch
* Autoscaling:
  + Add or remove servers based on alarms or resources increase/decrease
  + Aws autoscaling create-launch-configuration –launch-configuration-name my-lc –image-id ami-c6169af6 –instance-type t2.micro
  + Aws autoscaling create-auto-scaling-group –auto-scaling-group-name my-asg –launch-configuration-name my-lc –max-size 5 –min-size 1 –availablility-zones “us-west-2c”
  + Aws autoscaling put-scaling-policy –policy-name my-scaleout-policy –auto-scaling-group-name my-asg –scaling-adjustment 30 –adjustment-type PercentChangeInCapacity
    - Gives us the Policy ARN back
  + Aws autoscaling put-scaling-policy –policy-name my-scalein-policy –auto-scaling-group-name my-asg –scaling-adjustment 30 –adjustment-type ChangeInCapacity
    - Gives us the Policy ARN back
  + Cloudwatch Alarm – aws cloudwatch put-metric-alarm –alarm-name AddCapacity –metric-name CPUUtilization –namespace AWS/EC2 \ --statistic Average –period 60 –threshold 80 –comparison-operator GreaterThanOrEqualToThreshold \ --dimensions “Name=AutoScalingGroupName, Value=my-asg” –evaluation-periods 2 –alarm-actions <ARN>
  + Cloudwatch Alarm – aws cloudwatch put-metric-alarm –alarm-name RemoveCapacity –metric-name CPUUtilization –namespace AWS/EC2 \ --statistic Average –period 60 –threshold 40 –comparison-operator LessThanOrEqualToThreshold \ --dimensions “Name=AutoScalingGroupName, Value=my-asg” –evaluation-periods 2 –alarm-actions <ARN>
  + Sudo apt-get install stress
    - Stress –cpu 2 –timeout 600

AWS Logs

* Allows you to monitor your existing system, application, and custom logs in real time.
* You can send existing logs to CloudWatch, create patterns to look for in your logs, and alert yourself based on the findings of the patterns.
* Agent
  + Free for Windows, Amazon Linux, Ubuntu
* Purpose:
  + Monitor AWS CloudTrail logged events
  + Monitor Logs from EC2 instances in real-time
  + Archive log data
* **Log events** – a record sent to CloudWatch Logs to be stored. Timestamp and Message
* **Log Streams** – sequence of log events that share the same source. Automatically deleted after 2 months
  + Inherit expiration across groups
* **Log Groups** – groups of log streams that share the same retention, monitoring and access control settings
* **Metric Filters** – these are used to define how a service would extract metric observations from events and turn them into data points for CloudWatch metric. Assigned to log groups and log streams.
* **Wget** [**https**://s3.amazonaws.com/aws-cloudwatch/downloads/latest/awslogs-agent-setup.py](https://s3.amazonaws.com/aws-cloudwatch/downloads/latest/awslogs-agent-setup.py)
  + **Sudo python** ./awslogs-agent-setup.py –region us-west-2
* **Retention Settings** – How long log events are kept in CloudWatch Logs. Expired logs are automatically deleted

CloudWatch Log Filters

* **Filters** – define search patterns to look for in a log. These can then be turned into a metric and graphed.
  + Filters will **NOT** work on existing log data.
  + It will only work on data pushed to CloudWatch **AFTER** the filter was created
  + Only returns the first 50 results
* Metrics Contained:
  + Filter pattern
  + Metric Name
  + Metric Namespace
  + Metric Value
  + In order to make sure data is sent change /etc/rsyslog.d/50-default.conf and remove auth from it.

Real Time Log Processing

* Subscriptions feed logs into different services like capture, process, analyze.
  + Amazon Kinesis Streams
  + AWS Lambda
  + Amazon Kinesis Firehose
* Example
  + Locate commands in the resources directory
  + Export AWS\_DEFAULT\_REGION=us-west-2
  + Aws kinesis create-stream –stream-name “GuruLogs” –shard-count 1
  + Aws kinesis describe-stream –stream-name “GuruLogs
  + Write down your StreamARN
  + Aws iam create-role –role-name CWLtoKinesisRole –assume-role-policy-document <file://~/resources/TrustPolicy.json>
  + Write down your StreamARN
  + Put in the Permissions.json doc
  + Aws iam put-role-policy –role-name CWLtoKinesisRole –policy-name Permissions-Policy-For—CWL –policy-document <file://~/resources/Permissions.json>
  + We can now look to retrieve logs from kinesis and view logs
  + **Shard Iterator** – position in the stream from which the consumer will read from.
    - aws kinesis get-shard-iterator –stream-name GuruLogs –shard-id shardId-0000000000 –shard-iterator-type TRIM\_HORIZON

CloudTrail

* records all API calls made to any other AWS service and deliveries it to a log to you and records on behalf of a user by an AWS service
* Purpose is to enable security analysis, track changes, and provide compliance auditing
* **“InvokedBy”** field lets you see what triggered the API call
* **Log File validation –** uses SHA256 to make it hard to delete file, modify, or change the file.
* Logs contain:
  + identity of who made the API call
  + Time of the call
  + Source IP of the call
  + Request parameters
  + Response elements returned by the AWS service.
* Two types of Trails
  + All regions
  + One region
* Storage:
  + Store in S3 using SSE
  + Can store as long as you like or use lifecycle rules
  + Logs are delivered within 15 minutes of an API call
  + New logs are published every 5 minutes or so.

CloudWatch Events

* Similar to CloudTrail but faster. **Central Nervous System of AWS.** Near real-time stream of events.
* Components:
  + **Events (JSON)**
    - **State change** – when an AWS resource changes state, such as an EC2 instance changing from pending to running or when autoscaling creates or shuts down an instance.
    - **API call** – When an API call is made that is delivered to cloudwatch events via cloudtrial.
    - **Own code** –
  + **Rules**
    - Match incoming events and route them to one or more targets
    - They’re not ordered, and all rules that match an event will be processed
    - Rules can customize the JSON that flows to the target and elect to pass only certain keys and values.
    - Rules can specify multiple targets
  + **Targets**
    - Lambda Functions
    - Kinesis streams
    - SNS topics
    - Built-In

**Security/Governance/Validation:**

Intro

* Roles & Role architecture
* Cross Account Access – using roles
* Role Switching via the GUI and Command Line
* Identity Federation
* Web Identity Federation

Delegation & Federation

* **Delegation** – allow users in other AWS accounts, access to resources in yours directly
* **Federation** – allows users from external Identity Provider access to your account.
  + Types:
    - **Corporate/Enterprise Identity Federation**
      * Sources include – AD, LDAP
      * SAML, AWS Directory Service, Custom Federation Proxy
    - Web Identity Federation
      * Trust Amazon, FB, Google, Twitter and OpenID Connect
      * Used when you want to give an application access to your AWS resources
      * **Cognito** – handles this interaction
* **Roles** – is an object entity which is granted permissions to your account. contains two policy documents
  + **TRUST** Policy
  + **ACCESS** Policy
* **Sessions** – set of temporary credentials. Access and secret key with an expiration.
  + Obtained via STS – AssumeRole, AssumeRoleWithSAML, and AssumeRolewithWebIdentity
  + May or may not involve cognito
  + After temporary credentials expire, users have to go through it again
  + Services auto refresh the session which auto refreshes temporary credentials (Lambda or EC2). Managed on your behalf.

Corporate Identity Federation

* How?
* Allows you to allow to use an existing identity store for AWS
* Identity stores can be AWS Directory Services, SAML, or custom federation proxy.
* Temporary access given by STS GetFederationToken or AssumeRole
* Uses TRUST and ACCESS policies
* How it works?
  + STS provides you with session credentials, AKID, Secret Access Key, Session Token & Expiration Date
  + Expiration values are (Min/Max/Default)
  + AssumeRoke session 15 minutes, 1 hour, 1 hour
  + GetFederationToken 15minutes, 36 hours, 12 hours
  + Within AWS you have concept of an ‘identity provider’ its an IAM object which holds configuration information about the external identity providers
  + We generally map groups in your identity provider with Roles inside AWS accounts
* Why?
  + It allows the separation of responsibilities, your organization may have a dedicated identity team
  + You have one definitive source of identities within your business HR entry and exit processes are tied to this.
  + You minimize the admin overhead regarding identity management in the business.
  + You reduce the number of identities your staff need to manage and remember – reduced attack footprint.
  + **KNOW THE FLOW OF ASSUMEROLE**
  + **GETFEDERATIONTOKEN FLOW**
    - IAM user needs access to all Users with all permissions which is a risk. Needs a IAM user to take action
    - **GetFederation token doesn’t support MFA**
  + **ASSUMEROLEWITHSAML FLOW**
    - Don’t need to maintain a dedicated federation proxy
    - Proxy doesn’t need to hold any IAM policies

Web Identity Federation

* What?
  + Allows a trusted third party to authenticate users
  + Avoids us having to create and manage users in IAM
  + Avoids users having to remember multiple ID
  + Simplifies access control via roles
  + Improves security, no perm credentials stored in our application
  + Using Web Identity Federation + Cognito also provides user state syncing
  + Use if you want to publish an app with multiple credentials
* **Web Identity Federation FLOW**
* **Cognito** – identity management and sync services
  + **Cognito Identity –** web identity in AWS
  + **Cognito Sync –** application and user sync data across applications
  + **Cognito identity pool –** collection of identities, it allows grouping of identities from different providers as a single entity
  + Allows identities to persist across devices
  + Allows two roles to be associated, one for users authenticated by public idP or OpenID Providers
  + Second role can provide permissions for un-authenticated users.
  + **Pre-Cognito auth flow**
  + **Cognito Unauthenticated FLOW**
    - **Basic Classic (Non—Preferred) flow**
      * Same as before
    - **Advanced or Simplified flow**
      * Cognito handles the communication with STS versus before
  + Can orchestrate the generation of an unauthenticated identity
  + Can merge that identity into an authenticated identity if both are provided
  + Cognito can merge multiple identities into one object if all are provided
  + When ID’s are merged – any synced data is also merged.

**High Availability and Elasticity:**

Autoscaling

* Scale your EC2 instance capacity automatically according to your conditions
* Increase instances during demand spikes, maintain performance, decrease capacity during lulls
* Saves money
* Suitable for stable demand applications or hourly/daily/weekly demand fluctuations
* **Autoscaling Group** – instances are organized into groups so that they can be treated as a logical unit. When you create a group, you can specify its minimum, maximum, and desired number of EC2 instances. Limit is 20.
* **Launch Configuration** – Autoscaling Group uses a launch configuration as a template for its EC2 instances to specify what AMI will be launched, which keypair to use and what instance type. Limit is 100.
* **Scaling Plan** – tells your Auto Scaling Group how and when to scale. You can create scaling plans based on conditions (dynamic) or time (scheduled).
* Lifecycle hooks per AutoScaling group is 50
* Load balancers per Auto Scaling Group is 50 (10 attached at a time)
* Step adjustments per scaling policy is 20.
* Benefits
  + Better fault tolerance
  + Better availability
  + Better cost management

Autoscaling Lifecycle

* Refers to the life of an instance that is in a autoscaling group
* Starts when the Auto Scaling Group launches an instance
* End when **you** terminate the instance or Ends when the **Auto Scaling Group** takes the instance out of service and terminates it.
* AutoScaling Group -> Pending (Scale Out) -> Pending:Wait/Pending Proceed (EC2\_INSTANCE\_LAUNCHING) -> InService -> Terminating (Scale In/Fail Health Check) -> Terminate: Wait/Terminating Proceed (EC2\_INSTANCE\_TERMINATING) -> Terminated
* InService -> EnteringStandby -> Standby
* InService -> Detaching -> Detached -> EC2Instance

Autoscaling Lifecycle Hooks

* How it works?
  + Autoscaling responds to a scale out event by launching an instance
  + Autoscaling puts the instance into the Pending:Wait state
  + Autoscaling sends a message to the notification target defined for the hook, along with information and a token
  + Waits until you tell it to continue or the timeout ends
  + You can now perform your custom action, install software, etc
  + By default, the instance will wait for an hour and will change state to Pending:Proceed, then it will enter the InService state.
  + Can restart by changing heartbeat or changing the wait time
* Commands:
  + You can change the heartbeat timeout, or you can define it when you create the lifecycle hook in the CLI with the **heartbeat-timeout** parameter
  + You can call the **complete-lifecycle-action** command to tell the AutoScaling group to proceed
  + You can call the **record-lifecycle-action-heartbeat** command to add more time to the timeout.
  + 48 hours is the **maximum** time you can keep a server in a wait state, regardless of heartbeats.
* Cooldowns
  + Ensure the Autoscaling group doesn’t launch/terminate more instances than needed.
  + Cool down start when an instance enters the InService state, so if an instance is left in the Pending:Wait state as you perform functions on it, Autoscaling will still wait before adding any additional servers.
* **Abandon** – cause Autoscaling to terminate the instance and if necessary launch a new one
* **Continue** – put the instance into service.
* Spot Instances
  + Can **use** Lifecycle hooks with Spot instances
  + **Doesn’t** prevent an instance from terminating due to a change in the spot price
  + When a spot instance terminates, you must still complete the lifecycle action

Launch Configurations

* Template used by Autoscaling to launch EC2 instances.
  + Defines AMI ID, Instance Type, Key Pair, SG, Block device mapping
* **MUST** specify a launch configuration when creating a AutoScaling Group
* **Can** use a launch configuration with multiple Autoscaling groups
* **Can** only specify one launch configuration for a Autoscaling Group at one time
* **Can’t** modify a launch configuration after you’ve created it.
* Can create launch configurations from running EC2 instances
* Some properties may not be supported
* Overrides – AMI, block devices, key pair, instance profile, instance type, kernel, monitoring, placement tenancy, ramdisk, security groups, spot price, user data, assign public IP, EBS optimized.
* Can’t use the same launch configuration to launch on-demand instances and spot instances.
* How spot instances work
  + Set your bid price in your launch configuration
  + You need to create a new launch configuration to change your bid price
  + If your instance is terminated, Autoscaling will attempt to launch a replacement to maintain desired capacity.

AutoScaling Groups

* Contains a collection of EC2 instances that have similar characteristics, treated as a logical group
* Allows improved scaling and management of instances
* Can automatically create new servers when demand is high and remove servers when demand is low
* Process:
  + Launches EC2 instances to meet desired capacity
  + Maintains this number of instances by performing periodic health checks
  + If an instance is unhealthy, its terminated and a new instance is launched to replace it
  + Scaling policies are checked and the autoscaling group will adjust accordingly
* Before creating an Autoscaling Group:
  + Note the following:
    - How long it takes to launch and configure
    - What metrics have the most relevance to performance
    - How many Availability Zones you want to use
    - What role you want Autoscaling to play
    - What existing resources are used
* Can create an Autoscaling group by specifying an EC2 instance
  + Must also specify attributes such as minimum, maximum and desired instances
  + Autoscaling will automatically create your launch configuration and will associate it with your Autoscaling Group
  + Launch configuration will take the AMI ID, instance type and AZ from the EC2 instance.
* Limitations:
  + Tags aren’t copied
  + Autoscaling group includes the block device mapping from the AMI not the instance itself
  + Load balancer name is not copied to autoscaling **LoadBalancerNames** attribute.

Autoscaling Groups and Self Healing

* Used for:
  + Creating a low cost, self-healing, immutable infrastructure
  + No additional software to install or configure
  + Keep servers running and highly available without user interaction
* Good for:
  + Important servers you need to stay online, but only require one
  + Bastion/Jump box or an OpenVPN server
* Min 1. Desire 1. Max 1 to create self-healing HA
* Sudo pip install beeswithmachineguns paramiko
* Bees up –s 10 –g bees –k bees
* Bees attack –n 1000 –c 250 –u ELB

Amazon RDS

* RDS – makes it easy to create, operate, and scale databases. Their fast, cost efficient, resizable, secure, and highly available. Doesn’t require a lot of manual work.
  + Provisions infrastructure, automatic backups, automatic patching, install database software, automatic failover, synchronous data replication
  + You: Settings, schema, performance tuning
  + Vertical Scaling – db.t2.micro to db.r3.8xlarge. RI purchases of 1 to 3 years
    - Storage – 5 GB to 6 TB. Can scale live and can take a long time. **SQL Server will NOT let you scale your storage. Have to create a new database if you run out of storage.**
  + Horizontal Scaling – read replicas
    - Master-slave with great for high read to write ratios. 1 master that replicates to slaves. Up to 5 read replicas. Can promote masters. Can create replica of a replica.
    - Sharding – split tables into multiple databases. Split tables that aren’t joined by queries
* Available engines – MySQL, MariaDB, SQL, PstgreSQL, Oracle, Aurora
* Uses General Purpose SSD with 3 IOPS/GB bursting up to 3,000 IOPS or PIOPS which can specify your IOPS at creation. I/O optimized and varies by engine

Amazon Aurora

* Aurora – RDS database that provides 5 x throughput of MySQL on same hardware. Compatible with MySQL 5.6. Fault tolerant and self healing storage. Disk failures are repaired in background. Detects crashes and restarts. Fails over to 15 read replicas automatically.
  + No crash recovery or cache rebuilding
  + Autoscales from 10 GB to 64TB
* Backups:
  + Stored in S3
  + No impact on database performance
  + Up to 35 day retention period
  + PIT restore within a second
  + Automatic, continuous, incremental backups
* Snapshots
  + User-initiated snapshots are stored in S3
  + Kept until you explicitly delete them
  + Incremental
* Failure and Fault Tolerance
  + 6 copies of your data across 3 AZ
  + Recovery in an healthy AZ
  + Data divided into 10GB segments across many disks
  + Transparently handles loss
  + Can lose 2 copies of data without affecting write
  + Can lose 3 copies of data without affecting read
  + Storage is self-healing
* Replicas
  + Aurora
    - Can have up to 15
    - Low performance impact on primary
    - Replica can be a failover target with no data loss
    - Updates made by primary are visible to all replicas
    - Shares underlying volume with the primary instance
    - Replica can be a failover target with no data loss
  + MySQL Read
    - High performance impact on primary
    - Up to 5
    - Primary instance data is replayed on your replica as transactions
    - Replica can failover target with potentially minutes of data loss
* Security
  + SSL (AED-256) used to secure data in transit
  + Can encrypt DB using KMS
  + Encrypted Storage
  + Encrypted Backups
  + Encrypted Snapshots
  + Encrypted Replicas
  + **Note: You can’t encrypt an existing unencrypted database**

Dynamo DB Primer

* Fully managed, NoSQL Database. **Know NoSQL and SQL.** Scalability managed platform for performance
* No visible servers
* No practical storage limitations and fully resilient/HA
* Structure
  + DB is collection of tables which are the highest level of structures
  + 300 burst seconds
  + WCU (Write Capacity units) - 1 KB blocks per second
  + RCU (Read Capacity Units) – 4 KB blocks per second.
    - Doesn’t read from all locations
    - Eventually Consistent or Immediate consistent. 1KB or ½ KB
  + Schema or data structures is **not defined** at the table level.
  + Rows are items
  + Elements are attributes
  + **Hash/Partition** uniquely identifies an item. Helps find data by query function
  + **Sort/Range key** – allows a 1 to many relationship. Can provide a range of values
* Attributes
  + Dynamo DB Types
  + Scalar Types
  + String, Number, Binary, Boolean, Null, Document, Set
  + **Test DynamoDB. Look at Scan, Query, Batch and other operations.**

Dynamo DB Partitions

* They are the underlying storage and processing nodes of Dynamo DB
* 1 Table = 1 Partition
* All data initially is stored on one partition
* Partition can stored 10 GB and can handle 3000 RCU and 1000 WCU
  + New partition is added after these limits have gone over
* Data Distributed
  + Based on HASH/Partition key
  + 1 partitions key per partition. Each partitions can hold many keys
* they automatically increase. No automatic decrease when load/performance reduces
* Allocated WCU/RCU is split between partitions
* Be careful increasing and decreasing WCU/RCU
* Takeaways
  + Be aware of what influences the number of partitions
  + Be aware of the underlying storage infrastructure
  + Be aware that table performance is split across the partitions
  + Be aware that they increase, but don’t decrease
  + True performance is based on performance allocated, key structure, and time and key distribution of reads and writes

Dynamo DB Global and Secondary Indexes

* SCAN
* QUERY
* Without indexes, ability to retrieve information is limited to your primary table structure
* Global Secondary
* **Local Secondary (LSI)**
  + Only can be created at table creation
  + Allows us to maintain a secondary dataset which shares partition key but uses a different sort key
  + Contains partition, old sort key and new sort key (intrusion detection) + optional projected values
  + Data is asynchronous to any LSI
  + Shares RCU and WCU with a table
  + **LSI** is a Sparse index which will only have an ITEM if the index sort key is contained in table item.
  + Supports both eventually consistent reads and the other.
  + **Reads:**
    - Any non-key values by default aren’t in a LSI
    - If you query an attribute which is NOT projected, you are charged for the entire ITEM cost from pulling it from the main table. (Rounded up to 4K size)
    - If an attribute is not stored in the LSI, you experience latency and Dynamo DB fetches from the main table and get charged for it’s full cost of retrieving the main item which is rounded up to 4KB. Non-projected values.
    - Take care with planning your LSI and item projections
    - Main table contains 6 attributes which means it has 300 Bytes or so with rounded up to 4KB. Does a sub query. Sub queries total in this example about 20Kb which is 4 + 16. 4KB is also the query sizes
  + **Writes:**
    - Adding an item takes about 1 write per row
    - Deleting a row also costs 1 write unit.
    - Updates require 2 write units for delete and create.
  + Storage Concerns:
    - Be aware of Item Collections
    - Limits the size to 10GB
    - Only relate to tables with LSI’s.
    - **ItemCollectionSizeLimitExceededException**
* **Global Secondary Indexes (GSI)**
  + We can have an alternative Partition & sort key
  + LSI, you can choose keys only, a selection of attributes (include) or (all) attributes
  + RCU and WCU are defined on the GSI in the same way as the table. Different than LSI sharing performance
  + Changes are written GSI asynchronously
  + Can only support eventually consistent reads.
  + **http:docs.aws.amazon.com/amazondynamodb/latest/developerguide/SecondaryIndexes.html**

Dyanmo DB Streams & Replication

* **Stream** – ordered record of updates to a dynamo DB table.
  + It records changes to a table and stores those values for 24 hours.
  + Changes occur 1 and only 1.
  + All changes to the Table occur in near real-time
  + Data persists for 24 hours. Partition key is added for each row added.
    - Only thing you can’t see is what was done to the partition key is hard to see.
  + **KEYS\_ONLY** – key attributes are written to the stream
  + Streams can be configured with 4 **“views”**
  + **NEW\_IMAGE** – entire item POST update is written to the stream.
  + **OLD\_IMAGE** – entire item PRE the update is written to the stream.
  + **NEW\_AND\_OLD\_IMAGES** – pre and post operation state of the item is written to the stream allowing more complex comparison operations to be performed
  + Use Cases:
    - Replication
    - Triggers
    - Games or large distributed applications with users worldwide – using a multi-master database model. A synced set of tables operating worldwide.
    - DR – a database table in one AWS region, replicating to another AWS region for DR failover
    - Lambda function triggered when items are added to a dynamo DB stream, performing analytics on data.
    - Lambda function triggered when a new user signup happens on your web app and data is entered into an users table.
* **Replication** – not actually built in. Streams and kinesis are built using Cloudformation.
  + Create or select the table to be replicated
  + Apply the cloud formation stack and wait
  + Login to the replication console and create a replication group identify.
  + Wait until the process has completed, in the background additional checkpoint tables will be created.
  + **Know use cases of sterams**
  + **Know streams + lambda allow traditional DB style triggers will answer a number of questions.**
  + **Know the stream views**
  + [**http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Streams.CrossRegionRepl.html**](http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Streams.CrossRegionRepl.html)

Dyanmo DB Performance Deep Dive

* Partitions, Partitions, Partitions
* Two formulas
  + **Performance** – Partitions = Desired RCU/3000 RCU + Desired WCU/1000 WCU
  + **Size** – Partitions = Data Size in GB/10 GB
    - (6,7) = 7
* allocated reads and writes are distributed across partitions. We only get 4000 WCU if we read and write in parallel from all underlying partitions. Need a good solid partition key model.
* Key selection:
  + Attribute should have many distinct values
  + Attribute should have a uniform write pattern across all partition key values.
  + Attribute should have a uniform temporal write pattern across time.
  + If any of the above aren’t possible with an existing value – you should consider a synthetic/created/hybrid value.
  + You shouldn’t mix HOT and COLD key values within a table.
  + Example: voting use case. Key sharding with a random function ^n.
* GSI – Performance Considerations
  + GSI’s have their own RCU and WCU values and use alternative keys.
  + Writes could be throttled if there is not enough WCU across main and GSI.
* Write Leveling
  + You could rely on burst – you get 300 seconds of your RCU/WCU (Not good)
  + Change your application – force it to spread periodic batch writes over time.
  + SQS could be used as a managed write buffer.
* Read Levelings
  + Increase RCU could be dangerous.
  + Burst capacity won’t work
  + Static HTML would lessen user experience
  + Caching…
  + **Watch lesson again**