Annadale and Arlington

7035227476

7038233750

SERVICES COVERED

CODECOMMIT

CODEBUILD  
CODEDEPLOY

CICD

Continuous Integration Continuous Delivery Pipeline

Automating every part of the SDLC ( Software Delivery Life Cycle )

Continuous Integration

* Developers push the code to a code repository
* A testing/build server checks the code as soon as it's published
* The developer gets feedback about if the tests and checks have passed or failed

Finds bugs early, fixes bugs

Deliver faster as the code is tested

Deploy often

Within the pipeline we would do different steps like testing, compilation, QA/QC, packaging etc. Then it would deploy it to the different SDLC environment servers thereby eliminating the need for manual intervention to deploy these to each of the environments.

Parts of CI/CD pipeline

* Version Control Repository ( Eg. Git)
* Pipeline ( Orchestrator ) - Ties everything together
* Build Server
* Deployer

Key Principles of a CI/CD Pipelines

* Automation
* Everything as Code ( No Manual Adjustments ) - This means that the likelihood for human error is removed from the equation
* Test, Test, Test
* Consistency
* Integrate Frequently ( The quicker the release cycle the

Continuous Delivery

Ensure that the software can be released whenever needed

Ensures deployments happen often and are quick

Shift away from one release every three months to 5 releases a day

That usually means automated deployment

CD vs CD

* Continuous Delivery
  + Ability to deploy often using automation
  + May involve a manual step “approve” a deployment
  + The deployment itself is still automated and repeated
* Continuous Deployment
  + Full automation every code change is deployed all the way to production
  + No manual interventions or approvals.

Source Control Strategies

* Components of Version Control
* Version Control Strategies
* Repository - A centralized place where all components below are part of
* Branch - a collection of files that has its own version history
* Commit - a single change on a given branch
* Pull Requests - a request to merge one branch into another.

Trunk Based Development

All work is done on the master branch

When the code reaches an agreeable state - tag and release from that point

Pros:

* Very small changes
* Continuous code merges
* Increased delivery throughput

Cons:

* Many Tests.

GitHub Flow

All work is done on individual branches

Branches are then merged frequently

Pros:

* Master is always releasable
* Short Lived Branches

Cons:

* Master is not always upt to date
* Large Changes to Master

Git Flow

Features merge to develop

Release branches

Feature & Hotfix branches

Pros:

* Master is always releasable
* Strict Control

Cons:

* Master is not always up to date
* Large changes to master
* Complicated to enforce

Environmental branching

Branch per environment

Changes all to one branch

Graduate release to production

Pros:

* Each environment has its own branch
* Prod is always releasable

Cons:

* Prod is not always up to date
* Large changes to Prod
* Multiple merges per release

Code  
AWS CodeCommit

GitHub or 3rd party

Build / Test

AWS CodeBuild

Jenkins or 3rd party CI server

Deploy / Provision

AWS Elastic Beanstalk

AWS CodeDeploy

User Managed EC2 Instance Fleets via CF templates.

To Orchestrate the entire pipe is done through AWS CodePipeline

Code Commit

* Version Control:
* Is the ability to understand the various changes that happened to the code over time and possibly roll back
* All these are enabled by using a version control systems such as Git
* A git repository can live on one’s machine but it usually lives on a central online repository
* Benefits are:
  + Collaborate with other developers
  + Make sure the code is backed up somewhere
  + Make sure it’s fully viewable and auditable.

Code Commit

* Git repositories can be expensive
* The industry includes
  + GitHub: Free public repositories, paid privates ones
  + Bit bucket
  + GitLab
  + Etc
* And AWS CodeCommit
  + Private git repositories
  + No size limit on repositories ( scales seamlessly )
  + Fully managed, highly available
  + Code only in AWS Cloud account => Increases Security and Compliance
  + Secure
  + Integrated with other services within AWS

Your code privately held, and it is more secure, and in the same cloud

CodeCommit can use HTTPS and ssh to work with repositories, and root users cannot work with repositories and require users to interact with it.

Preventing users from pushing to a particular branch on code commit

Condition:

StringEqualsExists

codecommit:references

- “refs/heads/master”

AWS CodeCommit - Automation

* AWS Codebuild/AWS CodePipeline
* AWS Lambda
* Amazon SNS
* Anything!

Notification Rules

Two Types, Full or Basic, and the event that notifies to the SNS topic

Triggers, then can trigger a different service.

Amazon EventBridge – Used to be known as Cloudwatch Events, additionally you can use event bridge to listen to a specific cloudtrail log to trigger an event. So Event bridges has the most targets to utilize.

Code Commit Triggers and Notifications

Within source and settings, notifications

Notification Rule

Detail Type

Basic || Full

Events that trigger notifications

Sections

* Comments
  + On Commits
  + On Pull Requests
* Pull Requests
  + Source Updated
  + Created
  + Status Changed
  + Merged
* Branches and Tags
  + Created
  + Deleted
  + Updated

These triggers post to Amazon SNS Topic

– codestar?

Triggers

Trigger Name

Events

All Repository Events

Push To Existing Branch

Create Branch or Tag

Delete Branch or Tag

You can also set these triggers on specific branches as well.

CloudWatch Events

The Centerpiece of all devops automations

Rules:

Awscodestarnotifications rule

Under the hood cloudwatch is what makes the triggers in the codecommit tab work

Create a rule

Event Pattern || Schedule

Service Name → CodeCommit

Event Type → CodeCommit Repository State Change

ARN

Event Pattern Review

Source: What the service is

Detail-type: Events for the given source []

It will then show an example response object

Then choose a target for the cloudwatch event rule

Code Commit and Lambda

The amazon devops exam utilizes questions around a lot of the tutorials within the documentation

Within the lambda dash you can choose the lambda code commit trigger

AWS CodeCommit - Secure the Repository.

IAM - Explicit Denies override all allow statements, by default things are denied

Creating a Role, and then you can manage a users ability to interact with a specific branch within a repo as well.

Branch Protection is done through IAM

Approval Templates.

Automate and Integrate Testing.

* Types of Tests
* Automating Tests

Types:

* Linting ( not a test, but it is a way to get code checking )
* Unit Testing ( Testing a small piece a code )
* Functional Testing ( Testing a section of code )
* Integration Code ( Testing integration of the application )
* End-to-End Testing ( Full flow testing )
* Load Testing ( Testing the full load and scaling of tests )
* Chaos Testing ( Deleting random thing, and making sure things like disaster recovery and other things are working )

Local Tests

* Linting, Unit, and sometimes functional

Build Server

* Linting, unit, functional, integration, end to end, and load testing

CodeBuild Overview

Fully Managed Build Service

Alternative to other build tools such as jenkins

Continuous scaling ( no servers to manage 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 from code repository

Build instructions can be defined in code buildspec.yml file

Output logs to amazon s3 & AWS CloudWatch Logs

Metrics to monitor CodeBuild Statistics

Use Cloudwatch Events to detect failed builds and trigger notifications

Use CloudWatch Alarms to notify if you need ‘thresholds’ for failures

CloudWatch Events / AWS Lambda as a Glue

SNS Notifications

Create Code Build

Name

Description

Build Badge: AWS CodeBuild now supports the use of build badges, which provide an embeddable, dynamically generated image (*badge*) that displays the status of the latest build for a project. This image is accessible through a publicly available URL generated for your CodeBuild project. This allows anyone to view the status of a CodeBuild project. Build badges do not contain any security information, so they do not require authentication.

Source

Source Provider → Code Commit

Repository

– Reference Type ( Branch, Git Tag, Commit ID )

Environment

Environment Image ( DOCKER )

Managed Image || Custom

Service Role ( Allows code build to do what it needs to do )

Timeout – between 5 min and 8 hours, default 1 hour

Queued Timeout – Default is 8 hours

BuildSpec File

Buildspec.yml – default can be custom

Artificats → where to put the builds

Logs → CloudWatch ( optional ) important to read the logs of the build if something failed

BuildSpec.yml Deep Dive

Very important

Version → Tells codebuild how to interpret the file

Phases

Different phases

phases/\*/run-as – Tells code build to run the commands a specific linux user

phases/\*/on-failure – Specifies an action to take upon a failure within a phase ( ABORT || CONTINUE )

phases/\*/finally – Commands specified in a finally block ar run after commands within the command block. These will run even upon failures. – IMPORTANT if any command fails in a phase the phase will fail. This means if a command that is in the finally block fails even if the commands in the command block were successful the phase will still fail.

Allowed Build Phases

Install ( OPTIONAL )

– runtime versions → specify the versions of particular code → you can use env vars for this

– commands —> commands for codebuild to run

Pre\_build (OPTIONAL) Represents the commands, that codebuild should run before build – ex. Sign in to amazon ECR or install npm dependencies

– commands

Build (OPTIONAL) commands to run during the build

–commands

Post\_build (OPTIONAL) – Represents any commands to be run after the build ex pushing a build docker image to ECR

–commands

Phase Details

SUBMITTED

QUEUED

PROVISIONING  
DOWNLOAD\_SOURCE  
INSTALL

PRE\_BUILD

BUILD

POST\_BUILD

UPLOAD\_ARTIFACTS

FINALIZING

COMPLETED

Env:

Variables — key value pairs

Parameter-store – key value pairs ( probably from parameter store )

ENV\_NAME: “parameter/store/path”

Git-credential-helper : yes / no → private git repositories

Artifacts: → What is kept post build

Files: → Locations of the files

Name: → Artifact Name

Discard Paths

BaseDirectory

Secondary Artifacts

Caches for builds

Codebuild Security

* AWS IAM Access
* Artifact Encryption
  + Remember Encryption is two way hashing is one way.
  + Enabled by default
  + Can use specified KMS key
* AWS CodeBuild Role – Service Role

CodeBuild

Codebuild Environment Values

CodeBuild Automation

* Triggers
  + Frequency, specific times
  + Can specify specific version
  + Build Timeout
  + All Triggers are done in EventBridge
* Notifications
* Monitoring
* Logs

Environment Variables in Build Environments

AWS\_DEFUALT\_REGION – where the build runs

AWS\_REGION – where the

Build and Manage CI/CD Artifacts Securely

* Source
  + Authenticated git repos ( Most Secure)
  + Public git Repos ( Least Secure )
  + Amazon S3
* Network
  + Own network with Amazon VPC ( Yours )(Most Secure)(Most Expensive)
  + AWS network ( Amazon )
* Runtime
  + AWS Docker Images
  + Your own Docker Images
  + Environment is completely deleted after each build
* Storage
  + Amazon S3
    - AWS KMS encryption or Amazon encryption
  + Amazon ECR
  + Anything with an API
* Access
  + Amazon IAM
    - Amazon Codebuild
    - Amazon S3
    - Amazon ECR
  + Amazon S3 Bucket Policies
    - Amazon S3
  + AWS KMS Key:
    - AWS KMS Key policies

Deployment / Delivery Strategies

* All-at-Once
  + Downtime
  + Hard to rollback
  + Fast Deployment
* Rolling Deploy
  + Limited Blast Radius
  + Easy to Rollback
  + Slower deployment
  + App must support multiple live versions
* Blue-Green ( Red-Black )
  + All at once movement.
  + Any app can support
  + Easy Roll back
  + Fast Deployment
* Canary Release (A/B Release is pretty much the same )
  + Kind of like a hybrid of Blue-Green and Rolling Deployment
  + Limit the users exposed to bugs
  + Easy roll back
  + Slow Deployment

Implement Deployment Strategies

* All-at-Once
  + Upgrading the software directly on the running servers
* Rolling Deployment
  + Choose the percentage of users is sent to the new server
  + Building identical environments
  + Weighted DNS Routing
* Blue-Green
  + Build identical environment
  + DNS Routing or Amazon ECS target groups
* Canary Release
  + Build Identical Environments
  + Geographic DNS routing, feature toggle, user selection, a certain subset of users can access the new environment

CI/CD Tool Selection

* AWS CodePipeline ( Orchestrator for codecommit codebuild and codedeploy )
* Jenkins

– These tools are pretty much the same function, however they are different to utilize

AWS CodePipeline

* No managing of the server
* Built in actions
* Custom Actions
* IAM based authentication

Jenkins

* Self-Managed Infrastructure
* Master and Slave Nodes ( All self managed )
* Plugins
* Total Control

AWS CodeDeploy

-We want to deploy our application automatically to many EC2 instances

There are several ways to handle deployments using open source tools ( Ansible, Terraform Chef, PUppet, etc…)

We can us the managed service AWS CodeDeploy

Each EC2 Machine ( or on Premise Machine ) must be running the CodeDeploy Agent

The Agent is continuously polling AWS Code Deploy for work to do

CodeDeploy then sends an appspec.yml file

The application is pulled from the storage source

Then the EC2 will run the deployment instructions

Then the CodeDeploy Agent will report of the success or failure of deployment on the instance

EC2 instances are grouped by deployment group ( dev /test / prd )

Lots of flexibility to define any kind of deployments

CodeDeploy can be chained into code pipeline and use artifacts from there

Code deploy can reuse existing setup tools works with any application , auto scaling integration

Note Blue / Green only works with ec2 instances ( not on premise machines )

Support for AWS Lambda deployments,

Code deploy does not provision resources

The way code build works. You have a fleet of ec2 instances that will build the code, they are organized into deployment groups. The organization of them is done via tags. Upon setting up the instance with the CodeDeploy agent it will be able to be accessed for work. The appsec.yml will provide the instructions for how to interact with code deploy.

The files section of the appspec.yml file is only available on EC2/On Premise Deployments

Provides information to codedeploy about which files from the application revision should be installed on the instance during the deployments install event. This setion is required only if you are copying files from your revision to locations on the instance

Files:

* Source: source file location

Destination: destination file location

File\_exists\_behavior

You can assign multiple source and destination pairs. Paths mentioned are relative to where the appspec.yml file lives.

The File exists behavior an optional instruction explains to codedeploy how to handle files that exist in that location, disallow, overwrite, retain

Each of the deployment methods carry different mappings for the appspec, therefore it is important to note the differences between the different mappings

CodeDeploy is a deployment service that automates applications deployments to ec2, on-prem, lambda, or ecs

Code, lambda, web, configs, exec, packages, scripts, or ultimedai

Code deploy primary components

– application: a application you want to deploy

– Compute platform: ec2 / on prem, Lambda, ECS

– Deployment configuration: A set of deployment rules and deployment success and failure conditions used by code deploy during a deployment.

* Different deployment configurations for Lambda or ECS compute platforms:
* Canary – Traffic is shifted in two increments. You can shoes from predefined canary options that specify the percentage of traffic shifted to your updated lambda functions or ecs task set in the first increment and the interval in minute before remaining traffic is shifted in the second increment.
* Linear: Traffic is shifted in equal increments with an equal number of minutes between each increment you can choose from predefined linear options that specify the percentage of traffic shifted in each increment and the number of minutes between each increment
* All at once: all traffic is shifted from the original lambda function or task set all at once.

Deployment Group

A set of individual instances. A deployment grup contains individually tagged instances, Amazon EC2 instances in amazon EC2 auto scaling groups or both.

Deployment Types

A method used to make the latest applicatio revisionson instances in a deployment group.

–In place deployment: the application on each isnstance in the applicatio group is stopped, then the latest application revision is installed and the nev version of the application is started and validated you can use a load balance so that each instance is deregistered during its deployment and then restored to service after the deployment is completed. Only deploymentst that use ec2 or on premise compute platforms can use in place deployments

– Blue / Green Deployments

IAM instance profile is an iam role that you attach to your ec2 instances, this profile includes permissions to use s3 or github repos where apps are located

Revision is a version of an app a aws lambda deployment revision is either yaml or json formatted file that specifies info about the deployment.

An ec2 / or on prem deployment revision is an archive file that contains source content and an app specificatio file ( AppSpec.file )

Service Role is an IAM role that grants permissions to an AWS service so it can access AWS resources. The policies attached to the service role determine which services can be accessed an the actions it can perform.

– Needs

Read either the tags applied to the instances or the amazon ec2 auto scaling group names associated with the instances. This is needed because it enables codedeploy to identify instances to which it can deploy applications.

To perform operations on instances amazon ec2 auto scaling groups and elastic load balancing load balancers

To publish informationto SNS topics so that notifications can be sent when specified deployment or instance events to occur.

To retrieve info about cloudwatch alarms to setu palarms monitoring for deployments

Code Pipeline

Continuous delivery

Visual Workflow

Source: Code repo or artifact

Build: CodeBuild / Jenkins etc

Load Testing: 3rd party tools like new relic

Deploy: AWS code deploy / beanstalk / cfn / ecs etc:

Made of stages

Each stage can have sequential actions and / or parallel actions

Stages examples: Build / Test / Deploy / Load Test/ etc…

Manual approval can additional be defined at any stage.

Each pipeline stage can create artifacts

Artifacs are passed stored in s3 and passed on to the nex stage

1. Trigger
2. CodeCommit ( source )
3. CodeBuild ( then stores the artifacts in s3 )
4. Then deployed using CodeDeploy

CodePipeline Automation:

We can create notifications on a pipeline which can go to either an SNS or a chatbot ( slack )

Jenkins Overview

* Purpose
  + CI/CD pipeline tool to help automate and coordinate various jobs necessary for a deployment
* Amazon EC2 instance ( or server-based )
* Master-worker for job parallelization
* Server based model for deploying applications

AWS CodeDeploy

Application

* EC2, ECS, AWS Lambda
* Deployment Group
  + Service Role
* Triggers to react to specific events within the code deploy
* AppSpec?

DNS routing, how code is rolled out

– Rolling deployment to Lambda

– Complex blue green deployments to ECS or EC2

AWS CloudFormation

Infrastructure as Code

Currently we have been doing a lot of manual work

All this manual work will be very tough to reproduce:

* In another region
* In another AWS account
* Within the same regionif everything was deleted

Wouldn’t it be great if all our infrastructure was … code?

That code would be deployed and create / update / delete our infrastructure

What is CloudFormation

CloudFormmation is declarative way of outlining your AWS Infrastructrue, for any resources ( most of them are supported).

For example, within a CloudFormation template you say:

* I want a security group
* I want tow EC2 machines using this security group
* I want two Elastic IPs for these EC2 machines
* I want an S3 bucket
* I want a load balancer ( ELB ) in front of these machines

Then CloudFormation creates those for you in the right order, with the exact configuration that you specify

Infrastructure as code

* No resources are manually created which is excellent for control
* The code can be version controlled for example using git
* Changes to the infrastructure are reviewed through code

Cost

* Each resources within the stack is stagged with an identifier so you can easily see how much a stack costs you
* You can estimate the costs of your resources using the CloudFormation template
* Savings strategy: In dev you could automation selection of templates at 5 PM and recreated at 8 AM safely

Productivity

* Ability to destroy and recreate an infrastructure on the cloud on the fly
* Automated generation of Diagram for your templates
* Declarative programming ( no need to figure out orderingand orchestration )

Separation of concern:

* Create many stacks for many apps, and many layers. EX
* VPC stacks
* Network Stacks
* App Stacks

Don’t re-invent the wheel

Leverage web templates and the documentation

How Cloud Formation Works

* Templates have to be uploaded in S3 and then referenced in cloudformation
* To update a template we can’t edit previous ones. We have to reupload a new version of the template to AWS
* Stacks are identified by a name
* Deleting a stack deletes every single artifact that was created by CloudFormation

Deploying CloudFormation Templates

Manual Way:

Editing templates in the CloudFormation Designer

Using the console to input parameters etc

Automated way

* Editing templates in a YAML file
* Using the AWS CLI to deploy the templates
* Recommended way when you fully want to automate your flow

CloudFormation Building Blocks

Templates Components ( one course section for each )

1. Resources: the aws resources declared ( MANDATORY )
2. Parameters: dynamic inputs for your template
3. Mappings: static variables for the template
4. Outputs: References to what has been created
5. Conditionals: List of conditions to perform resource creation
6. Metedata

Template Helpers

1. References
2. Functions

CloudFormation supports YAML and JSON

CloudFormation Parameters

Parameters are a way to provide inputs to your AWS CloudFormation templates

They’re important to know about if:

* You want to reuse your templates across the company
* Some inputs can not be determined ahead of time

Parameters are extremely powerful controlled and can prevent errors from happening in your templates thanks to types.

When should you use a parameter

* Is this cloudfromation resource configuration likely to change in the future?
* If so then make it a parameter

You won’t have to reupload a template to change its content

Parameters:

SecurityGroupDescription:

Description: Security Group Description ( Simple Parameter)

Type: String

Parameters Settings

Parameters can be controlle by all these Settings

Type:

* String
* Number
* CommaDelimitedList
* List<Type>
* AWS Parameter ( to help catch invalid values – match against existing values in the AWS Account )
* Description
* Constraints
* ContraintDescription (String)
* Min/Max Length
* Min/Max Value
* Defaults
* AllowedValues ( array )
* AllowedPattern ( regexp )
* NoEcho ( Boolean ) – This is to make sure not to print any secrets.

How to Reference a Parameter

* The Fn:Ref function can be leveraged to reference parameters
* Parameters can be used anywhere in a template.
* The shorthand for this in YAML is !Ref
* The function can also reference other elements within the template
* The !Ref can be used to reference other resources

DbSubnet1:

Type: AWS::EC2::Subnet

Properties:

VpcId: !Ref MyVPC

Concept: Pseudo Parameters

* AWS offers us pseudo parameters in any CF template
* These can be used at any time and are enabled by default.

AWS::AccountId

AWS::NotficationARNs

AWS::NoValue

AWS::Region

AWS::StackId

AWS::StackName

What are Resources

* Resources are the core of you CF template ( mandatory )
* They represent the different AWS Components that will be created and configured
* Resources are declared and can reference eah other
* AWS figures out creation, updates, and deletes of resources for us
* There are over 224 types of resources
* Resource types identifiers are of the form
  + AWS::aws-product-name::data-type-name

What are mappings?

* Mappings are fixed variables within your CloudFormation Template
* They’re very handy to differentiate between different environments ( dev vs prod ), regions ( AWS regions), AMI types, etc
* All the values are hardcoded within the template

Mappings:

Mapping01:

Key01:

Name: Value01

Mapping02:

Key02:

Name: Value02

When would you use mappings vs parameters?

* Mappings are great when you know in advance all the values that can be taken and that they can be deduced from variables such as
  + Region
  + Availability Zone
  + AWS Account
  + Environment ( dev vs prod )
  + ETc
* They allow for safer control over the template
* Use Parameters id you need the values to be really user specific

Fn::FindInMap

Accessing Mapping Values

* We use Fn::FindInMap to return a named value from a specific key
* !FindInMap [ MapName, TopLevelKey, SecondLevelKey ]

What are outputs?

* The outputs section declares optional output values that we can import into other stacks ( if you export tehm first )!
* You can also view the outputs in the AWS Console or in using the AWS CLI
* They’re very useful for example if you defined a network CloudFormation, and output the variables such as VPC ID and your Subnet IDs
* It’s the best way to perform some collaboration cross stack, as you let expert handle their own part of the stack.
* You can’t delete a CloudFormation Stack if its outputs are being referenced by another CloudFormation stack

Outputs Example

* Creating a SSH Security Group as part of one template
* We create an output that references that security group

Outputs:

StackSSHSEcurityGroup:

Description: The SSH Security Group for our Company

Value: !Ref MyCompanyWideSSHSecurityGroup

Export:

Name: SSHSecurityGroup

Cross Stack Reference

* We then create a second template that leverages that security group
* For this we use the Fn::ImportValue function
* You can’t delete the underlying stack until all the references are deleted too.

Resources:

MySecureInstance:

Type: AWS::EC2::Instance

Properties:

AvailabilityZone: us-east-1

ImageId: ami-a4c7edb2

Instancetype: t2.micro

SecurityGroups:

- !ImportValue SSHSecurityGroup

What are conditions used for?

* Conditions are used to control the creation of resources or outputs based on a condition
* Conditions can be whatever you want them to be, but common ones are:
  + Environment ( dev / test / prod )
  + AWS Region
  + Any parameter
* Each condition can reference another condition parameter value or mapping

How to define a condition?

Conditions:

CreateProdResources: !Equals [ !Ref EnvType, prod ]

The logical ID is for you to choose. It’s how you name the condition

The intrinsic function ( logical ) can be any of the following:

* Fn::And
* Fn::Equals
* Fn::If
* Fn::Not
* Fn::Or

Using a Condition

* Conditions can be applied to resources / outputs / etc…

Resources:

MountPoint:

Type: “AWS::EC2::VolumeAttachment”

Condition: CreateProdResources

CloudFormation Must Know Intrinsic Functions

* Ref
* FN::GetAtt
* Fn::FindInMap
* Fn::ImportValue
* Fn::Join
* Fn::Sub
* Condition Functions (Fn::If, Fn::Not, Fn::Equals, etc… )

Fn::Ref

* The Fn::Ref function can be leveraged to reference
  + Parameters => returns the value of the parameter
  + Resources => returns the physical ID of the underlying resource ( ex: EC2 ID)
* The shorthand for this in YAML is !Ref

DbSubnet1:

Type: AWS::EC2::Subnet

Properties:

VpcId: !Ref MyVPC

Fn::GetAtt

* Attributes are attached to any resources you create
* To know the attributes of your resources, the best place to look at is the documentation.
* For example: the AZ of an EC2 machine!

Resources:

EC2Instance:

Type: “AWS::EC2::Instance”

Properties:

Type: “AWS::EC2::Instance”

Properties:

ImageId: ami-1234567

InstanceType: t2.micro

NewVolume:

Type: “AWS::EC2::Volume:

Condition: CreateProdResources

Properties:

Size: 100

AvailabilityZone:

!GetAtt EC2Instance.AvailabilityZone

Fn::FindInMap

Accessing Mapping Values

* WE use Fn::FindInMap to return a named value from a specific key
* !FindInMap [ MapName, TopLevelKey, SecondLevelKey ]

Fn::ImportValue

* Import values that are exported in other templates
* For this , we use the Fn::ImportValue function

Resources:

MySecureInstance:

Type: AWS::EC2::Instance

Properties:

AvailabilityZone: us-east-1a

ImageId: ami-a4c7edb2

InstanceType: t2.micro

SecurityGroups:

* !ImportValue SSHSecurityGroup

Fn::Join

* Join values with a delimiter
* !Join [ delimiter, [ comm-deliminated list of values ] ]
* This creates “a:b:c”
* !Join [ “:” , [ a, b, c ] ]

Function Fn::Sub

* Fn::Sub, or !Sub as a shorthand, is used to substitute variables from a text. It’s a very handy function that will allow you to fully customize your templates.
* For example, you can combine Fn::Sub with References or AWS Pseudo variables
* String must contain ${VariableName} and will substitute them

!Sub

* String
* { Var1Name: Var1Value, Var2Name: Var2Value }

!Sub String

Condition Functions

* The logical ID is for you to choose. It’s how you name condition
* The intrinsic function ( logical ) can be any of the following
  + Fn::And
  + Fn::Equals
  + Fn::If
  + Fn::Not
  + Fn::Or

CloudFormation User Data

User Data in EC2 for CloudFormation

* We can have user data at EC2 instance launch through the console
* We can also include it in CloudFormation
* The important thing to pass is the entire script through the function Fn::Base64
* Good to know: user data script log is in /var/log/cloud-init-output.log

Cfn-init

* AWS::CloudFormation::Init must be in the Metadata of a resource
* With the cfn-init script it helps make complex ec2 configurations readable
* The Ec2 instance will query the CloudFormation service to get init data
* Logs go to /var/log/cfn-init.log
* Let’s see how it works through a simple CloudFormation

CloudFormation Service Launches the Ec2 instances

The EC2 instances retrieves the init data from the CFN service and then runs the cfn-init script

If something fails within the cfn init script the instance will still be creation completed in status. To cause an error you need to use the cfn-signal and wait conditions

We still don’t know how to tell CloudFromation that the EC2 instance got properly configured after a cfn-init script

* For this, we can us the cfn signal script
  + We run cfn signal right after cfn init
  + Telling cloudformation service to keep on going or fail
* We need to define the WaitCondition
  + Block the template until it receives a signal from teh cfn signal
  + we then attach a CreationPolicy ( Also works on EC2 AutoScaling Groups )

Wait Condition Didn’t Receive the Required Number of Signals from an Amazon EC2 Instance

Ensure that the AMI you’re using has the AWS CloudFromation helper scripts installed. IF the AMI doesn’t include the helper scripts you can also download them to your instance

Verify that the cfn-init & cfn-signal command was successfully run on the instance You can view logs, such as /var/log/cloud-init.log or /var/log/cfn-init.log, to help debug the instance launch.

You can retrieve the logs by logging in to your instance, but you must disable rollback on failure or else AWS CloudFormation will delete the instance after the stacks fail to create

Verify that the instance has a connection to the Internet. If the instance is in a VPC the instance should be able to connect to the internet through a NAT device if it’s in a private subnet or through an Internet Gateway if it’s in a public subnet

Rollbacks on failures

Stack Creation Fails: (CreateStack API)

* Default: Everything rolls back ( gets deleted ). We can look at the log
  + OnFailure=ROLLBACK
* Troubleshoot: Option to disable rollback and manually troubleshoot
  + OnFailure=DO\_NOTHING
* Delete: Get rid of the stack entirely do not keep anything
  + OnFailure= DELETE

Stack Update Fails ( UpdateStack API )

* The stack automatically rolls back to the previous known working state
* Ability to see in teh log what happened and error messages

NESTED STACKS

Nested stacks are stacks as part of other stacks

They allow you to isolate repeated patterns / common components in separate stacks and call them from other stacks

Example:

Load Balancer configuration that is re-used

Security Group that is re-used

Nested stacks are considered best practice

To update a nested stack always update the parent ( root stack )

ChangeSets

* When you update a stack, you need to know what changes before it happens for greater confidence
* ChangeSets won’t say if the update will be successful
* It will tell you which resources will be updated, added, or deleted

Retaining Data on Deletes

* You can put a Deletion Policy on any resource to control what happens with the CloudFormation Template is deleted.

DeletionPolicy=Retain

* Specify on resources to preserve/back up in case of CloudFormation deletes
* To keep a resource specify Retain ( works for any resource / nested stack )

DeletionPolicy=Snapshot:

* EBS Volume, ElastiCache Cluster, ElastiCache ReplicationGroup
* RDS DBInstance, RDS DBCluster, Redshift Cluster

DeletePolicy=Delete ( Default Behaviour ):

* Note: for AWS::RDS::DBCluster resources, the default policy is Snapshot
* Note: to delete an S3 bucket, you need to first empty the bucket of its contents

Termination Protection on Stacks

* To prevent accidental deletes of CloudFormation templates use Termination Protection

Parameters in Cloudformation

* Getting parameters from SSM
  + Parameters > paramName
    - Type: ‘AWS::SSM::Parameter::Value<String>
    - Default: /parameter/file/path/in/ssm
* Public Parameter Heirarchy:
  + Type: ‘AWS::SSM::Parameter::Value<AWS::EC2::Image::Id>’
  + Default: ‘/aws/service/ami-amazon-linux-latest/amzn2-ami-hvm-x86\_64-gp2’

Obtaining a list of amazon managed parameter store values can be done through the aws cli

aws ssm get-parameters-by-path –path /aws

DependsOn:

Is a way to say that this resource should not be created till the listed resource in the DependsOn indicates.

Lambda Functions

Inline reference:

* ExecutionRole
* Handler - indicates what class/object contains the method for the lambda function
* Code:
  + ZipFile: |
  + This allows you to create a literal string of code, however it is limited to 4000 characters and cannot use dependencies that are not already found in the aws runtime environment.

Zip File Reference

* Code:
  + S3Bucket: bucket-name
  + S3Key: key-parameter-path-to-file-and-file-name
  + S3ObjectVersion:
    - Reference the version of an item if versioning is enabled for the s3 bucket

CloudFormation Custom Resources ( Lambda )

* You can define a Custom Resource in Cloudformation to address any of these use-cases:
  + An AWS resource is yet not covered ( new service for example )
  + An on-premise resource
  + Emptying an S3 bucket before being deleted
  + Fetch an AMI id
  + Anything you want

CloudFormation Custom Resource ( Create, update, delete )

Calls an AWS Lambda Function

API calls -> whatever you want

The lambda function will get invoked only if there is a create, update or delete an event, not every time you run the template

Drift Detection

How to detect if something has changed from the cloudfromation setup ie. there was some manual changes to resources created by cloudformation.

Example:

LambdaUsedToCleanUp:

Type: Custom::cleanupbucket

Properties:

ServiceToken: !ImportValue EmptyS3BucketLambda ( What lambda to invoke with the events)

BucketName: !Ref myBucketResource

CAPABILITY\_IAM or CAPABILITY\_NAMED\_IAM

– You need to provide the cloudformation the ability to create IAM Roles

* InsufficientCapabilitiesException Error, It means that you have forgotten to provide cloudformation the ability to provision certain resources.

Cfn-hup

The cfn-hup helper is a daemon that detects changes in resource metadata and runs user-specified actions when a change is detected. This allows you to make configuration updates on your running Amazon EC2 instances through the UpdateStack API action.

Cfn-hup.conf file:

Stores the name of the stack and the aws credentials that the cfn-hup daemon targets

* Stack
* Credential-file
* Role
* Region
* Unmask
* Interval – The interval at which the daemon checks the metadata
* Verbose – verbose logging

Hooks.conf configuration file:

The user actions that the cfn-hup daemon calls are defined here.

Hookname

Triggers, post.add, post.update, post.update

Path: – the path to the metadata object

Action: – Arbitrary shell command that is run as given

Runas: – A user to run the command as. Cfn-hup uses the su command to switch to the user.

Stack Policies:

A stack policy is a JSON document that defines the update actions that can be performed on designated resources. Upon setting a stack policy, all resources in the stack are protected. ONe can allow updates on specific resource by specifying an explicit Allow statement for the resource within the stack policy.

You can only define one policy per stack. You additionally cannot associate different stack polices with different users.

The stack policy only apples to stack updates. It does not provide access controls like AWS IAM policy.

The stack policy should only be used as a fail-safe mechanism to prevent accidental updates to specific stack resources.

This policy is an example of how to prevent updates to the ProductionDatabase resource

{

"Statement" : [

{

"Effect" : "Allow",

"Action" : "Update:\*",

"Principal": "\*",

"Resource" : "\*"

},

{

"Effect" : "Deny",

"Action" : "Update:\*",

"Principal": "\*",

"Resource" : "LogicalResourceId/ProductionDatabase"

}

]

}

As stated above when you apply a stack policy to a stack all resources have to follow the policy by default. By telling all resources to allow updates, than explicitly deny the update for a specific resource you do a dependency inversion. Meaning you do not have to specify ALL the different resources and allow them.

There is a risk in using a default denial. If you have an ALlow statement elsewhere in the policy. YOu might unknowingly grand update permission to resources that you don’t intend to. Because an explicit denial overrides any allow actions you can ensure that a resource is protected using a Deny statement.

Elastic Beanstalk

EB CLI

Elastic Beanstalk Command Line Interface

aws-cli-elastic-beanstalk -scripts

.elasticbeanstalk file

k

**CodeCommit**

* <https://www.atlassian.com/git/tutorials/using-branches>
* <https://docs.aws.amazon.com/codecommit/latest/userguide/auth-and-access-control-iam-identity-based-access-control.html>
* <https://aws.amazon.com/blogs/devops/refining-access-to-branches-in-aws-codecommit/>
* <https://docs.aws.amazon.com/codecommit/latest/userguide/how-to-notify.html>
* <https://docs.aws.amazon.com/codecommit/latest/userguide/how-to-repository-email.html> )
* <https://docs.aws.amazon.com/codecommit/latest/userguide/how-to-notify-lambda.html>
* <https://docs.aws.amazon.com/codecommit/latest/userguide/how-to-migrate-repository-existing.html>

**CodeBuild**

* <https://docs.aws.amazon.com/codebuild/latest/userguide/build-spec-ref.html>
* <https://docs.aws.amazon.com/codebuild/latest/userguide/samples.html>
* <https://docs.aws.amazon.com/codebuild/latest/userguide/sample-docker.html>
* <https://aws.amazon.com/blogs/devops/validating-aws-codecommit-pull-requests-with-aws-codebuild-and-aws-lambda/>

**CodeDeploy**

* <https://docs.aws.amazon.com/codedeploy/latest/APIReference/API_MinimumHealthyHosts.html>
* <https://docs.aws.amazon.com/codedeploy/latest/userguide/reference-appspec-file-structure-hooks.html>
* <https://docs.aws.amazon.com/codedeploy/latest/userguide/reference-appspec-file-structure-hooks.html#appspec-hooks-server>
* <https://docs.amazonaws.cn/en_us/codedeploy/latest/userguide/reference-appspec-file-structure-hooks.html#reference-appspec-file-structure-environment-variable-availability>
* <https://docs.aws.amazon.com/codedeploy/latest/userguide/monitoring-cloudwatch-events.html>
* <https://aws.amazon.com/blogs/devops/view-aws-codedeploy-logs-in-amazon-cloudwatch-console/>
* <https://docs.aws.amazon.com/codedeploy/latest/userguide/monitoring-sns-event-notifications.html>
* <https://docs.aws.amazon.com/codedeploy/latest/userguide/deployments-rollback-and-redeploy.html>
* <https://docs.aws.amazon.com/codedeploy/latest/userguide/deployment-groups-configure-advanced-options.html>
* <https://docs.aws.amazon.com/codedeploy/latest/userguide/instances-on-premises.html>
* <https://docs.aws.amazon.com/codedeploy/latest/userguide/register-on-premises-instance-iam-user-arn.html>
* <https://docs.aws.amazon.com/codedeploy/latest/userguide/register-on-premises-instance-iam-session-arn.html>
* <https://docs.aws.amazon.com/codedeploy/latest/userguide/deployment-configurations.html#deployment-configuration-lambda>
* <https://docs.aws.amazon.com/codedeploy/latest/userguide/reference-appspec-file-structure-hooks.html#appspec-hooks-lambda>

**CodePipeline**

* <https://docs.aws.amazon.com/codepipeline/latest/userguide/reference-pipeline-structure.html#action-requirements>
* <https://docs.aws.amazon.com/codepipeline/latest/userguide/best-practices.html#use-cases>
* <https://docs.aws.amazon.com/codepipeline/latest/userguide/actions-invoke-lambda-function.html>
* <https://docs.aws.amazon.com/codepipeline/latest/userguide/actions-create-custom-action.html>
* <https://docs.aws.amazon.com/codepipeline/latest/APIReference/API_PutJobSuccessResult.html>
* <https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/continuous-delivery-codepipeline.html>
* <https://docs.aws.amazon.com/codepipeline/latest/userguide/tutorials-cloudformation.html>
* <https://github.com/aws-samples/codepipeline-nested-cfn>
* <https://aws.amazon.com/blogs/devops/implementing-gitflow-using-aws-codepipeline-aws-codecommit-aws-codebuild-and-aws-codedeploy/>

**CodeStar**

* <https://docs.aws.amazon.com/codestar/latest/userguide/templates.html>

**Jenkins**

* <https://aws.amazon.com/getting-started/projects/setup-jenkins-build-server/>
* <https://wiki.jenkins.io/display/JENKINS/Amazon+EC2+Plugin>
* <https://aws.amazon.com/blogs/devops/setting-up-a-ci-cd-pipeline-by-integrating-jenkins-with-aws-codebuild-and-aws-codedeploy/>
* <https://wiki.jenkins.io/display/JENKINS/AWS+CodeBuild+Plugin>
* <https://wiki.jenkins.io/display/JENKINS/Amazon+EC2+Container+Service+Plugin>
* <https://wiki.jenkins.io/display/JENKINS/Artifact+Manager+S3+Plugin>
* <https://wiki.jenkins.io/display/JENKINS/AWS+CodePipeline+Plugin>

#### **Reference Links for Domain 2**

I use the documentation a lot in this course. Here are all the links that are visited during this section. I recommended you read through them in your own time, as I judge them to have some importance for your understanding of the services and the exam. Happy reading!

**CloudFormation:**

* <https://stackoverflow.com/a/45007029/3019499>
* <https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/walkthrough-custom-resources-lambda-lookup-amiids.html>
* <https://github.com/awslabs/aws-cloudformation-templates/tree/master/aws/solutions/lambda-backed-cloudformation-custom-resources/Fetch-AMI-From-Parameter-Store>
* <https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/using-cfn-describing-stacks.html#w2ab1c15c15c17c11>
* <https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/troubleshooting.html#troubleshooting-errors-update-rollback-failed>
* <https://aws.amazon.com/blogs/devops/continue-rolling-back-an-update-for-aws-cloudformation-stacks-in-the-update_rollback_failed-state/>
* <https://stackoverflow.com/a/41468341/3019499>
* <https://docs.aws.amazon.com/AWSCloudFormation/latest/APIReference/API_CreateStack.html#API_CreateStack_RequestParameters>
* <https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/using-iam-template.html#using-iam-capabilities>
* <https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/protect-stack-resources.html>

**Beanstalk**

* <https://github.com/aws/aws-elastic-beanstalk-cli-setup>
* <https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/eb-cli3-install-advanced.html>
* <https://aws.amazon.com/blogs/devops/using-the-elastic-beanstalk-eb-cli-to-create-manage-and-share-environment-configuration/>
* <https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/command-options-general.html#command-options-general-autoscalingasg>
* h[ttps://docs.aws.amazon.com/elasticbeanstalk/latest/dg/ebextensions-optionsettings.html](https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/ebextensions-optionsettings.html)
* <https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/environment-resources.html>
* <https://stackoverflow.com/a/40096352/3019499>
* <https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/create_deploy_docker_v2config.html>

**Lambda:**

* <https://docs.aws.amazon.com/serverless-application-model/latest/developerguide/serverless-sam-cli-install.html>
* <https://docs.aws.amazon.com/serverless-application-model/latest/developerguide/automating-updates-to-serverless-apps.html>

**Step Functions**

* <https://aws.amazon.com/step-functions/use-cases/>

**API Gateway**

* <https://aws.amazon.com/blogs/compute/introducing-amazon-api-gateway-private-endpoints/>
* <https://docs.aws.amazon.com/apigateway/latest/developerguide/api-gateway-request-throttling.html>
* <https://docs.aws.amazon.com/step-functions/latest/dg/tutorial-api-gateway.html>

**ECS**

* <https://docs.aws.amazon.com/AmazonECS/latest/developerguide/service-auto-scaling.html>
* <https://docs.aws.amazon.com/AmazonECS/latest/developerguide/using_awslogs.html>
* <https://docs.aws.amazon.com/AmazonECS/latest/developerguide/using_cloudwatch_logs.html>
* <https://ecsworkshop.com/introduction/cicd/>
* <https://docs.aws.amazon.com/AmazonECS/latest/developerguide/ecs-cd-pipeline.html>
* <https://docs.aws.amazon.com/codepipeline/latest/userguide/tutorials-ecs-ecr-codedeploy.html>
* <https://aws.amazon.com/blogs/devops/build-a-continuous-delivery-pipeline-for-your-container-images-with-amazon-ecr-as-source/>

**Opsworks**

* <https://docs.aws.amazon.com/opsworks/latest/userguide/welcome_classic.html>
* <https://docs.aws.amazon.com/opsworks/latest/userguide/workingcookbook-events.html>
* <https://docs.aws.amazon.com/opsworks/latest/userguide/workingcookbook-json.html>

White Papers

Whitepaper Review Containerized Microservices

* Microservices
* Containers
* Organization Structures
* Product Not Projects
* Smart Endpoints Dump Pipes
* Design for Failure

Microservices are an architectural and organizational approach to software development in which software is composed of small, independent services tat communicate over well-defined APIs

* Decoupled
* Single Function
* Well-Defined API’s
* 12-Factor App
  + Isolate Dependencies
  + Ephemeral ( Disposable )
  + Scalability ( Concurrency )

Organization Structures

* Autonomous Teams
* Own Microservice all the way to production

Product Not Projects

* Automation
* Self-Service
* Continuous Integration
* Continuous Delivery

Smart Endpoints Dump Pipes

* Communication between services
  + Request/Response
  + Publish/Subscribe

Decentralized Governance

* Freedom to choose dependencies
* Freedom to choose build tools

Decentralized Data Management

* Freedom to choose data tools
* No shared data sources

Design for Failure

* Disposable ( stateless )
* Latency and Timeouts to be expected
* Able to failover to other region
* Self-healing
* Logs as event streams

Deployment Services

* AWS CodeDeploy
* Amazon ECS

AWS CodeDeploy

* Service that can deploy applications to
  + Amazon EC2
  + On-premise
  + Amazon ECS (including AWS Fargate)
  + AWS Lambda
* Rolling and Blue/Green Deployments
* Integrates with AWS CodePipeline and other CI/CD tools

Code Deploy is more complicated to setup than other deployment services. You will additionally need the code deploy agent installed on the different servers. EC2 and On-Premise compute servers work well with code deploy the best

Amazon ECS

You can do deployments without AWS CodeDeploy but only for Amazon ECS

* Only for Amazon ECS ( including AWS Fargate
* Rolling Update
  + Launches new tasks to the cluster and then switches over the traffic to the new tasks
* Circuit Breaker available through CLI/API/SDK only ( not in the console )
  + As it goes through the tests there is a check that will check the conditions if it fails it will rollback to the previous tasks immediately.

ECS tends to be much simpler than code deploy. Code deploy is better for advanced setups, but ECS is better for Docker and simpler deployments.

Infrastructure Deployment Concepts

* Pets Vs. Cattle
* IaaS, PaaS, and SaaS
* Serverless Infrastructure

How you treat your deployment and servers

Pets

* Have names ( live a long time )
* Fixed when broken
* Hard to Maintain

Cattle

* Short Lived
* Replaced when broken
* Maintained with code

As a service

* Infrastructure as a Service (IaaS) - Eg EC2 instance
  + The vendor takes on
    - Hardware
  + We control
    - Operating System
    - Application
* Platform as a Service (PaaS) - Eg Elastic Beanstalk
  + The vendor takes on
    - Hardware
    - Operating System
  + We control
    - Application
* Software as a Service (SaaS) - CodeBuild, CodePipeline
  + Vendor takes on
    - Application
    - Operating System
    - Hardware

Serverless Infrastructure - You do not manage the server Eg. Lambda or Fargate

* Servers are still being used
* No management of underlying compute power
* Runs when you need it

Security Concepts in the Automation of Resource Provisioning

* Configuration Management
* Patch Management
* Security Compliance

Configuration Management

* Consistency
  + All of the servers should be the same
* Auditability
  + Easily audit changes that occur to a configuration
* Versioning
  + Easy to version a configuration to see changes over time and provide the ability to rollback to a previous configuration.

Making a unified way of managing all of your instances

Vulnerability Management

* Monitoring for vulnerabilities
* Fixing vulnerabilities
* AWS Services to help:
  + AWS Systems Manager
  + AWS OpsWorks

Security Compliance

* Set of rules or best practices
* AWS Services that help:
  + AWS Systems Manager
  + AWS Lambda
  + AWS Config

Implement Lifecycle Hooks on a Deployment

* Lifecycle Hooks are wait-states for AWS Auto Scaling Group
* Use Cases:
  + Configuration
  + Health Checks

Lifecycle Hooks on AWS AutoScaling Groups:

* Action for starting new instance and stopping instances
* Supports spot instances
* Cool Down is paused while hooks are running
* Health Check grace period starts AFTER Lifecycle Hooks.

Scale Out

* Pending
* Pending:Wait

– Do anything we need to do, then send a signal to proceed

* Pending:Proceed
* InService

Scale In

* Terminating
* Terminating:Wait
  + Do anything we need to do then send a signal to proceed
* Terminating:Proceed
* Terminated

Additionally there is a timeout that when hit will auto proceed with the process

Heartbeats

* Heartbeats prevent timeouts from occurring
* Maximum time a instance can be in a wait state is 48 hours or 100 times the heartbeat timeout, whichever is smaller
* Default timeout is an hour

Comparison of AWS Configuration Management Tools

* AWS CloudFormation
* AWS OpsWorks Stacks
* AWS OpsWorks for Chef Automate
* AWS OpsWorks for Puppet Enterprise

AWS CloudFormation

* Amazon EC2 Metadata
  + Interject configuration setup at runtime to run scripts at runtime
* CloudInit
  + Additional linux, declarative, add cookbook configuration of things.
* Imperative/Declarative
  + Imperative step by step
  + Declarative you use an abstracted solution to do basic setup
* Infrastructure as Code

AWS OpsWorks Stacks

* Chef Solo - Imperative ( all of the servers contain chef )
  + Deploying bash scripts to all of the instances
* Scaling
* Clustering
* Automation

AWS OpsWorks Chef Automate - A managed chef server

* Chef Automate - Imperative
  + Holds all of the cookbooks and recipes
  + Then all the clients get the updates from the main server
* Chef server PaaS

AWS OpsWorks for Puppet Enterprise

* Puppet Enterprise - Declarative
* Puppet enterprise Server PaaS
* Client Server Model

Cloudformation is only at start up, where chef and puppet will correct drift of long running servers. So Cloudfromation is more for cattle thought process where puppet and chef is for pet servers

Cloudformation - Overview

* What is CloudFormation
  + Create Infrastructure from code
  + Repeatable infrastructure - You can put it into a code version system ( git )
  + Update configurations with code
  + Create new or importing resources
  + Drift is not prevented
* Templates
  + Template Section
    - Format version
    - Description
    - Metadata
    - Parameters
    - Mappings
    - Conditions
    - Transform
    - Resources – Only Required Section
    - Outputs
  + YAML or JSON
* StackSets
  + Deploy stacks into multiple accounts
  + Regional
* Macros
  + Pre-processing of templates
  + Usually ran with AWS Lambda
* Custom Resources
  + Custom provisioning logic
  + Allows to use services that may not be supported by AWS CloudFormation yet
  + Best hosted on AWS Lambda
* CloudFormation Registry
  + Extensions for AWS CloudFormation
  + Public or Private
  + Anyone can create them

AWS OpsWorks - Overview

* AWS OpsWorks Stacks
  + Stacks
    - An application / environment a container
    - Linux and Windows
    - Chef Solo
    - On-Premise support
    - Lifecycle events - trigger changes to happen
    - Amazon IAM - control who has access to stacks
  + Layers
    - FE, BE, Database
  + Automate and orchestrates configuration changes
  + Scaling and monitoring
  + Customizable
* AWS OpsWorks for Chef Automate
  + Setus up Chef Automate server
  + Backups
  + Community developed tools or cookbooks
  + Native Chef tools ( KNife, ChefDK)
  + Control Network for Chef Server
  + Automatically registers instances with OpsWorks from ASG
  + Can SSH to Chef Automate Server
    - Best practice is to lock it down rather than have it open
* AWS OpsWorks for Puppet Enterprise
  + Setups up puppet enterprise master server
  + Backups
  + Community developed modules and manifests
  + Automatically registers instances with OpsWorks from ASG
  + Control Network for Puppet Server
  + Can SSH to Puppet Master Server
    - Best practice is to lock it down rather than have it open

AWS Elastic Beanstalk - Overview

What it is:

* Simplifies deployment of web applications
* Handles updates to the underlying resources
* Scaling and monitoring
* Deployment of new app versions
* Customizable

Important Details

* Supported programming languages: Java, .NET, Node, PHP, Ruby, Python, Go, and Docker
* Linux or Windows
* VPC support
* IAM control access

AWS Elastic Container Service - Overview

What is:

* Schedules docker containers to run on hosts
* Manages the running of a container
* Scheduled, and constantly running
* AWS Fargate vs AWS EC2 Launch Type
* Fargate takes care of managing the hosts as well which means all we need to worry about is the containers

Components

* Images - Amazon Elastic Container Registry
  + A singular container
* Containers
  + Multiple containers
  + What containers need to be next to each other ( sidecar containers )
  + Overarching capacity
* Task Definitions
  + Configuration file
* Service
* Cluster
  + Grouping of services
* Capacity Providers - AWS EC2/ASG

How to Deploy to AWS ECS

* AWS Console
* CLI/API
* AWS Cloudformation
* AWS CodeDeploy

Twelve Factor App Pattern

High Level Objectives

* Methodology for building Software as a service
* Minimize effort for contributors to learn
* Portability, Scalability, Consistency
* Built for Continuous Deployment
* Built for Cloud Deployments
* Codebase
  + Unified and centralized repository for code for all deploys
  + A deploy is a single deployment ( dev, staging, and prod would be three deploys )
  + All of these should be versioned
* Dependencies
  + Declare packages specifically; no assumptions
  + This means that everything that you need to run the app should be bundled together. You don’t assume that something is present for example the java runtime.
* Config
  + Separate configuration from code; Use environment variables
* Backing Services
  + All services should be resources; able to be replaced without change to code.
* Build Release Run
  + Strict separation
    - Build = dependencies + compile,
    - Release = app + config,
    - Run = execution of the release
* Processes
  + App runs a stateless process and shares nothing
* Port Binding
  + Export service to specific port
* Concurrency
  + Utilize processes to handle different workload types (eg http requests vs scheduled jobs )
* Disposability
  + Started / stopped any time
  + Handle shutdowns gracefully
  + Minimize startup time
* Dev / Prod Parity
  + Deploy code quickly
  + Same developers who write the code deploys it
  + Dev and prod are as similar as possible
* Logs
  + Sends logs to STDOUT; i.e. Don’t manage log files
* Admin Processes
  + Run one-off process as its own process

AWS Lambda Overview

* Concepts, Features, Quotas
* Security
* Monitoring and Logging

Concepts ( Function as a Service, A singular function )

* Serverless - No servers needed to manage
* Triggers
* Runtime - Everything except for your application ( function )
* Deployment Package
* Concurrency

Features

* Version and Alias
  + An alias points to a version
* Concurrency limits
  + Lambdas concurrency per AWS accounts
  + Limit the number of instances can be running in parallel
* Layers - Custom runtimes
  + Good for shared tools, or dependencies that are too large to bundle
* Step Functions
* Extensions

Quotas

* Memory - 1024MB
* Timeout - 15 minutes 900 seconds
  + If longer running than this ECS / Fargate would be where you would want to look
* Deployment Package - 50 MB Zipped; 250MB Unzipped
* /tmp storage - 512MB

Security

* IAM
  + Execution Role
  + User Access
* Data
  + Encryption in Transit (HTTPS)
  + Encryption at Rest
    - Secrets Manager or SSM Parameter Store for Environment Variables
  + Environment Variable SSM/Secrets Manager
* Monitoring
  + Cloudwatch
  + Lambda Insights (enhanced monitoring)
* Logs
  + Cloudwatch Logs
  + Custom Extensions ( for example something like splunk )

AWS Cloudformation Templates

Parameters – On start these values you will need to fill out to deploy them

ParameterName

Type: dataType ( string, number)

Default: defaultValue

AllowedValues: AllowedValuesToUse

Description: TheDescriptionOfTheParameter

Mappings

MapValues:

Case:

Variable: ValueToUse

Resources:

ResourceName

Type: WhatItIs

Properties:

Whatever properties is supported by the type

Tags:

Key: Value

Outputs:

Valuetoexport:

Value: !REF ‘valuetoref’

Cloudformation - Security

* AWS IAM
* AWS Cloudformation Service Role
* Logging API calls to AWS CloudTrail

Cloudformation - EC2 Initialization

UserData:

Fn::Base64: !Sub |

Bash scripts insert here, these are good for short scripts

AWS::CloudFormation::Init:

Config:

Packages:

AWS Cloudformation - Best Practices

* Organization/Preparation
  + Ownership and Lifecycle organization
  + Export shared resources to be imported
  + IAM to control access
  + Plan to reuse templates and write them
  + Verify quotas before you deploy
  + Nested Stacks to reuse common patterns
* Creating Templates
  + No credentials in templates
  + Use the AWS parameter types
  + Use parameter constraints
  + CFN init for deploying software to Amazon EC2 instances
  + Use the latest helper scripts for aws-cfn-bootstrap
  + Validate template before launching
* Managing Stacks
  + No manual updates to AWS Cloudformation resources
  + Change sets before updating
  + Stack policies
    - Prevents people from changing resources created from amazon cloudformation
  + Use AWS CloudTrail for auditing AWS Cloudformation
  + Version and peer-review templates

AWS OpsWorks - Technical Aspects

Chef Automate

* Workflow
  + Coordinate development, test, and deployment
  + Quality gates to promote code
  + Supporting collaboration between teams
    - Each team can implement its own gates
* Compliance
  + Configuration management
  + Reports for compliance
    - Center for Information Security ( CIS ) profile
* Visibility
  + Dashboards
  + Query Events
  + View related information about events

AWS OpsWorks - Recipe Anatomy

* Common Chef Resources
* Sample Recipe

Common Chef Resources

* Bash
  + Execute a script using the bash interpreter
* Directory
  + Manage directories
* Execute
  + Execute a single command
* File
  + Manage Files
* Git
  + Manage source resources in Git repositories
* Group
  + Manage groups
* Package
  + Manage packages
* Route
  + Manage a linux route table entry
* Service
  + Manage a service
* User
  + Manage users

OpsWorks Chef Automate

Best Practices

* Recipe Best Practices
* Storage and DR ( Disaster Recovery )
* Security

Recipe Creation

* Recipe Linting
  + Rubocop
  + Foodcritic
* Recipe Testing
  + Unit Testing - ChefSpec
  + Integration Testing - Test Kitchen

Storage and DR

* AWS S3 Storage for Recipes
* Backup Stragegies ( Cross Replication across regions )

Security

* AWS IAM for securing OpsWorks for Chef Automate API calls
* AWS CloudTrail for Logging

AWS ElasticBeanStalk

Limitations

* Deployments are difficult to troubleshoot
* No control or insights as to what is included in an update
* Slower deployments
* Not good for complex deployments
* No insight into the platform layer
* Not good for daily deployments

Customizations

* Multiple Resources
  + Amazon EC2 Instance
  + Security Group
  + Load Balancer
  + Load Balancer Security Group
  + Auto Scaling Group
  + Amazon S3 Bucket
  + Amazon CloudWatch Alarms
  + Amazon Cloudformation Stack
  + Domain Name
  + Custom Resources
* .eb Extensions
  + We can customize the previously listed resources through the .eb extensions
  + Set application options ( E.g Java Version )
* Env.yml
  + AWS ElasticBeanStalk Environment Configuration
  + Configuration for each environment
  + Can link Environments

Overview of Docker

* Containers
  + OS Level virtualization
  + Isolation by default
* Docker
  + Tool to build, deploy and run containers

Containers

Virtual Machines

Host - Server

Hypervisor - Multiple OS on a Server

OS and APPs

– The OS is duplicated

Containers

Host - server

OS

Container Scheduler – Docker

APPs ( Is the actual Container )

Docker

Tool to build containers

Agent that will run the containers

Tool to also deploy containers

Storage

* Storage of containers ( ECR or Docker Hub )
* Similar to git ( versioned )
* Centralized so CI/CD pipelines can all reach it

Elastic Container Service - ( ECR )

Lifecycle Policy - Prevent from having too many versions of a container.

AWS Elastic Container Service

Fargate

Task Definitions are static, and cannot be changed hence definition

With fargate you have to set the task memory and the task cpu

Task memory being the amount of memory used by the task

Task CPU the number of cpu units used by the tasks

AWS Elastic Container Service - Cloudwatch

Container Insights for ECS containers

What is Docker

Docker is a software development platform to deploy apps

Apps are packaged in containers that can be run on any OS

Apps run the same, regardless of where they’re run

* Any machine
* No comaptibility issues
* Predicttable behavior
* Less work
* Easier to maintain and deploy
* Works with any language as well

On one EC2 instance there can be multiple docker containers.

Additionally you can scale multiple same containers on the same instances

Where Docker images are stored

DOcker images are stored in Docker Repositories

Docker Hub

Private Docker repos ECR

Docker vs Virtual Machines

Docker is sort of a virtualization technology but not exactly

Resources are shared with the host allowing for many containers on one server

The big thing is that docker can share an OS where as VMs need their own operating systems

Docker Containers Management

To manage containers we need a container management platform

* ECS
* Fargate
* EKS

ECS Cluster Overview

ECS Clusters are logical grouping of EC2 instances

EC2 instances run the ECS agent Docker container

The ECS agents registers the instance to the ECS cluster

The EC2 instances run a special AMI, made specifically for ECS

Cluster Template

Networking Only ( AWS Fargate )

* Cluster
* VPC ( optional )
* Subnets ( optional )

EC2 Linux + Networking

* Cluster
* VPC
* Subnets
* Auto Scaling group with Linux AMI

EC2 Windows + Networking

* Cluster
* VPC
* Subnets
* Auto Scaling group with Windows AMI

EC2 + Networking

Provisioning Model

* On-Demand Instance
* Spot Instances

EC2 instance type

Number of Instances

EC2 AMI – The specific amazon EC2 ami id

EBS storage

Key Pair

Networking

VPC– Select a already created VPC

Subnets - The subnets where the cluster will deploy the EC2 AMI  
Security Group - Select a already created security group or create a new one

Container Instance IAM role

The Amazon ECS container agent makes calls to Amazon ECS API actions on your behalf, so container instances that urn the agent require the ecsInstanceRole IAM policy and role for the service to know that the agent belongs to you if you do not have the ecsInstanceRole already amazon can create one for you

Services

Tasks

ECS Instances

* You can see all the ecs instances here including the status and particulars of a given ec2 instance

Metrics

Scheduled Tasks

Tags

ECS Task Definitions

Task definitions are metadata in JSON form to tell ECS how to run a Docker COntainer

It contains crucial information around

* Image Name
* Port Binding for container and host
* Memory and cpu required
* Environment variables
* Networking information

Additionally the ECS Agent is just another Docker Container

For an application you can have the host port and then additionally the container port so when traffic from the internet comes in on a specific port EG our host port 8080 docker will know where to send the information due to the container port which could be 80

Select a launch type compatibility

Fargate

* Price based on task size
* Requires network mode awsvpc
* AWS managed infrastructure, no amazon ec2 instances to manage

EC2

* Price based on resource usage
* Multiple network modes available
* Self managed infrastructure using amazon ec2 instances

Task Definition Name - Arbitrary

Task Role - They additionally have a AWS IAM role

* Optional IAM role that tasks can use to make API requests to authorized AWS services. Create an Amazon Elastic Container Service Task Role in the IAM console

Network Mode

* If you choose <default> ECS will start your container using Docker’s default networking mode, which is Bridge on Linux and NAT on Windows. <default> is the only supported mode on Windows

Task Execution Role

This IAM role is required by tasks to pull container images and publish container logos to Amazon CloudWatch on your behalf. If you don't have the ecsTaskExecutionRole already, amazon can create one for you

Task Size

MIB

CPU

The task will take a portion of the memory and cpu of the container that it is on ( this can be important when determining how many different tasks can run on a ec2 unit in the cluster. )

Add a container

Name

Image location

* Imagename:versiontag

Port Mappings

Host Port, Container Port

ECS Service

* ECS Services help define how many tasks should run and how they should be run
* They ensure that the number of tasks desired is running across our fleet of EC2 instances
* They can be linked to ELB / NLB / ALB if needed
  + TLDR
  + ALB
    - Layer 7 (HTTP/HTTPS traffic), Flexible
    - This is the distribution of requests based on multiple variables form the network layer to the application layer. It is context-aware and can direct requests based on any single variable as easily as it can a combination of variables. Applications are load balanced based on their particular behavior and not solely on server ( operating system or virtualization layer) information
    - This is feature fulled Layer 7 load balancer, with only http and https listeners only. It provides the ability to route HTTP and HTTPS traffic based upon rules, host based or path based. Like an NLB each target can be on different ports. This additionally supports HTTP/2. There is additionally a configurable range of health check status codes ( CLB only supports 200 OK for HTTP health checks )
    - With an Application Load Balancer it is a requirement that you enable at least two more availability Zones
  + NLB
    - Layer 4 (TLS/TCP/UDP traffic), Static IPs
    - This is the distribution of traffic based on network variables such as IP address and destination ports. It is layer 4 (TCP) and below and is not designed to take into consideration anything at the application layer such as content type, cookie data, custom headers, user location, or the application behavior. It is context-less caring only about hte network layer information contained withinthe packets it is directing this way and that
    - This is a TCP Load Balancer only that does some NAT magic at the VPC level. It uses Elastic IPS, so it has a static endpoint unlike ALB and CLBs ( by default ). Each target can be on different ports
    - With NLB Elastic Load Balancing creates a network interface for each AZ that you enable.
  + CLB
    - Layer 4/7 (HTTP/TCP/SSL traffic) is Legacy and recommended to avoid using it
  + The application load balancer and network load balancer are designed from teh ground up for moder paradigm of dynamic port configurations as commonly seen in containerized deployments.
  + Key Differences
    - Application Load Balancer works at the application layer. Where as the NLB works at the Transport layer. NLB just forwards requests whereas ALB examines the contents of the HTTP request header to determine where to route the request. So, application load balancer is performing content based routing.
    - NLB cannot assure availability of the application. This is because it bases its decisions solely on network and TCP layer variables and has no awareness of the application at all. Generally a NLB determines availability based on the ability of a server to respond to ICMP ping, or to correctly complete the three way TCP handshake. ALB goes much deeper, and is capable of determining availability based on not only a successful HTTP Get of a particular page but als the verification that the content is as was expected based on the input parameters
    - When considering the deployment of multiple applications on the same host sharing ip addresses( virtual hosts in old school speak ) NLB will not differentiate between application a and application B when checking availability ( indeed it cannot unless ports are different ) but ALB will differentiate between the two applications by examining the application layer data available to it. This difference means the NLB may end up sending requests to an application that has crashed or is offline but ALB will never make that mistake.

**Amazon White Papers**

**Running Containerized Microservices on AWS**

Characteristics of a microservices architecture

* Componentization via services
* Organized around business capabilities
* Products not projects
* Smart endpoints and dumb pipes
* Decentralized governance
* Infrastructure automation
* Design for Failure
* Evolutionary Design

Twelve Factors Application

1. Codebase - one codebase tracked in revision control many deploys
2. Dependencies - explicitly declare and isolate dependencies
3. Config Store configurations in the environment
4. Backing Services - Treat backing services as attached resources
5. Build, Release, Run - Strictly separate build and run stages
6. Processes - Execute the app as one or more stateless processes
7. Port Binding - Export services via port binding
8. Concurrency - Scale out via the process model
9. Disposability - Maximize robustness with fast startup and graceful shutdown
10. Dev / Prod Pairty - keep development, staging and production as similar as possible
11. Logs - Treat logs as event streams
12. Admin processes - Run admin / management tasks as one-off processes

Componentization via Services

Key factors from the twelve-factor app pattern methodology that play a role in compentization:

* Dependencies ( explicitly declare and isolate dependencies ) - Dependencies are self-contained within the container and not shared with other services
* Disposability ( maximize robustness with fast start up and graceful shutdown ) Disposability is leveraged and satisfied by containers that are easily pulled from a repository and discarded when they stop running
* Concurrency ( scale out via the process model ) – Concurrency consists of tasks or pods ( made or containers working together ) that can be auto scaled in and out in a memory - and CPU efficient manner

Each business function is implemented as its own service

Each service should have its own integration and its own deployment pipeline.

Coordination Layer that tracks which containers are running on the different hosts.

Additionally a system that monitors the state of containers, the resources available in a cluster, etc.

Amazon ECS refers to the container management system as tasks

Kubernetes refers to them as Pods

Schedulers maintain the desired count of container set for the service.

Organized Around Business Capabilities

Boundaries, Is an application a microservice, is a shared library a microservice are team dependent.

Before microservices, system architecture would be organized around technological capabilities such as user interface, database, and server-side logic. In a microservices-based approach, as a best practice, each developmentteam owns the lifescycle of its service all the way to teh customer.

In a microservices-driven organization, small teams act autonomously to build, deploy, and manage code in production. This allows teams to work at their own pace to deliver features. Responsidbility and accountability foster a culture of ownership, allowing teams to better align to the goals of their organization and be more productive.

Microservices are as much an organizational attitude as a technological approach this principle is known as Conway’s Law:

“Organizations which design systems … are constrained to produce designs which are copies of the communication structures of these organization”

When architectures and capabilities are organized around atomic business functions, dependencies between components are loosely coupled. As long as there is communication contract between services and teams each team can run at its own speed. Then with this approach the stack can be polyglot, meaning that developers are free to use the programming languages and technology that are optimal for their component.

Meaning that business functions can drive development decisions. Organizing around capabilities means that each API team owns teh function, data, and performance completely.

Key factors from the 12 factor app that play a role in organizing around capabilities.

* Codebase - Each microservice owns its own codebase in a separate repository and throughout the lifecycle of the code change
* Build Release Run - Each microservice has its own deployment pipeline and deployment frequency this enables the development teams to run microservices at varying speeds so they can be responsive to customer needs.
* Processes - Each microservice does one thing and does that one thing really well. The microservice is designed to solve the problem at hand in the best possible manner.
* Admin Processes - Each microservice has its own administrative or management tasks so that it functions as designed.

Popular Microservice Design Patterns

* Aggregator Pattern - A basic service which invokes other services to gather the required information or achieve the required functionality. This is beneficial when you need an output by combing data from multiple microservices
* API Gateway Design Pattern - API Gateway also acts as the entry point for all the microservices and creates fine-grained APIs for different type of clients. It can fan out the same request to multiple microservicess and similarly aggregate the results from multiple microservices.
* Chained or Chain of Responsibility Pattern - Chained or Chain of Responsibilites Design Patterns produces a single output which is a combination of multiple chained outputs.
* Asynchronous Messaging Design Pattern - In this type of microservice design pattern all the services can communicate with each other, but they do not have to communicate with each other sequentially and they usually communicate asynchronously
* Database or Shared Data Pattern - This design pattern will enable you to use a database per service and a shared database per service to solve various problems. These problems can include duplication of data and inconsistency, different services have different kinds of storage requirements, few business transactions can query the data and with multiple services and denormalization of data
* Event Sourcing Design Pattern - This design pattern helps you to create events according to change of your applicationstate.
* Command Query Responsibility Segregator (CQRS) Design Pattern - This design pattern enables you to divide the command and query. Using the common CQRS patternwhere the command part will handle all the requests related CRUD while the query part will take care of the materialized views
* Circuit Breaker Pattern - This design pattern enables you to stop the process of the request and response when the service is not working. For example when you need to redirect the request to a different service after certain number of faile request intents.
* Decomposition Design Pattern - This design pattern enables you to decompose an application based on business capability or on based on the subdomains.

====================================================================

# 1 / 30 / 2023

ECS Task Definitions

* Tasks definitions are metadata in JSON form to tell ECS how to run a Docker Container
* It contains crucial information around:
  + Image Name
  + Port Binding for Container and HOst
  + Memory and CPU required
  + Environment variables
  + Netoworking information

Container port the port theapplication is running the Host port is the host is exposing eg

ECS Service with Load Balance

In the ECS task definition you leave the host port blank which will cause ECS to choose a random port. Then with an application load balancer it will forward traffic to the needed clusters with its feature of dynamic port forwarding

You can only add a load balancer on service creation

To allow the ALB to communicate with any port on the containers you will need to update the inbound rules on the security group. Then allow all traffic on any port to a custom resources and select the security group for the ALB.

ECR

* So far we’ve been using Docker images from Docker Hub (public)
* ECR is a private Docker image repository
* Access is controlled through IAM ( Permissions errors => policy )

Example commands

* $(aws ecr get-login –no-include-email –region eu-west-1)
* Docker push 1234567890.dkr.ecr.eu-west-1.amazonaws.com/demo:latest
* Docker pull 1234567890.dkr.ecr.eu-west-1.amazonaws.com/demo:latest

Fargate:

When launching an ECS Cluster we have to create our EC2 instances

If we need to scale we need to add EC2 instances

So we manage infrastructure

With Fargate its all Serverless

We don’t provisionEC2 instances

We just create task definitions, and AWS will run our containers for us

To scale just increase the task number super simple, and no more EC2 management ( still using ec2 under the hood )

Create clusters then create task definitions

Create cluster, create task definitions, create service

Minimum healthy percent set to 0 will allow for rolling deployments

Elastic Beanstalk and ECS

YOu can run elastic beanstalk in single and multi docker container mode

Multi docker helps run multiple containers per EC2 instance in EB

This will create for you

ECS Cluster

EC2 instances, configured to use the ECS Cluster

Load Balancer ( in high availability mode )

Task definitions and executions

Requires a config file Dockerrun.aws.json at the root of the source code

Your docker images must be pre built and stored in ECR for example

ECS IAM roles

Ecs instance role

This allows the ec2 instance too interact with the ecs api

Task Role

Ecs task and it provides the docker container the ability to interact with aws resources≥

Two types of IAM roles for ECS

* ECS Instance Role - AmazonEC2ContainerServiceforEC2Role
* ECS Task Definition has a Task Role - Is a role that can only be assigned to ECS tasks and this will provide the docker container to do things to AWS for example interacting with an S3 bucket.

Autoscaling for ECS

Service Auto Scaling

* Minimum number of tasks
* Desired number of tasks
* Maximum number of tasks
* IAM role for Service Auto Scaling - it will use the ecsAutoscaleRole

Then add a autoscaling policy

* Target tracking
  + Track a service metric
  + Then assign a target value to scale the service at a specific metric value
  + Scale out cooldown
  + Scale in cooldown
* Step Scaling
  + Execute the policy when
  + Set a cloudwatch alarm to trigger the scaling
  + Very similar to the auto scaling group triggers for EC2 instances

Tricky bit with autoscaling classic ecs clusters.

The above autoscaling policy will handle the ecs container instances however it does not scale the underlying ec2 instances up or down. We need then link up the auto scaling group with the cluster autoscaling policy

This is hard to do because you have to define two autoscaling policies one for the ECS service and the other for the EC2 instances

For Fargate this is far easier since you only need to create a autoscaling policy for the ECS Service.

Both have autoscaling at the Service level

Elastic Beanstalk creates an easier way to autoscaling between the ec2 and ecs instances with the multidocker EB setup

Integrations between ECS and Cloudwatch

Logs:

You can define a log driver, or a log configuration, then you can define the specific the cloudwatch logs for the ecs. By setting this up will allow the docker container to send logs to the cloudwatch log group. The big thing to also do is to make sure the ecs task definition has the correct permissions to send logs to cloudwatch.

Additionally you can configure a loging configuration for the ecs daemon for the

Cloudwatch Metrics

You can look at the different metrics for different clusters within ECS, and this under hood can be used for the autoscaling

Cloudwatch Container insights - If enabled you can get the metrics for all the individual metrics for each container. However it does cost extra to enable this feature.

Cloudwatch Events:

State changes:

Specific detail type: ECS Container instances state change and ECS task state change

Then specify all cluster or a specific cluster allowing us to react to state changes of ECS.

CI/CD Pipeline for ECS

ECSworkshop.com

CodeDeploy for Blue Green Deployment for ECS

Since latest is a mutable tag it will take the latest image of the ECR container.

Image Tag or Digest Sha 56 can be used for versioning for the right task definition docker container.

AWS OpsWorks Stacks

OpsWorks Stacks

OpsWorks for Chef Automate

OpsWorks for Puppet Enterprise

OpsWorks is using chef cookbooks for their deployments.

A stack is a set of layers instances and related AWS resources whose configuration you want to manage together

For an example of the diagram of how this works it can be seen:

docs.aws.amazon.com/opsworks/latest/userguide/welcome\_classic.html

Users interact with the internet that traffic reaches the AWS Cloud, inside our VPC we have our OpsWorks Stack, inside the stack we have a few different layers including our load balancer, our application server layer, and our amazon rds layer.

Cookbook are pieces of code in some sort of specific formats to tell the application layer ( which is managed by us ) on how to provision itself.

We will need to tell OpsWorks where the cookbooks are located ( git, s3, archive etc )

Layers within OpsWorks are a blueprint for a set of amazon ec2 instances. It specifies the instance settings, associated resources, installed packages, profiles, and security groups. YOu can also add recipes to lifecycle events of your instances for example: to set up deploy configure your instance or discover your resources.

Auto Healing Enabled. If a instance is unhealthy opsworks will attempt to redeploy to heal itself.

There are different types of instances for example we have time based instances and load balanced instances.

The time based instance you can define an instance and add a schedule to them to tell them at what time and day of the week the instance should be spun up.

Load Based Instance which can be spun up when a specific load target has been surpassed.

So the three types are

24/7

Time

Load

This provides a autoscaling solution but it is not as flexible as autoscaling groups

Apps

Shows the different apps configured and can then be deployed on to the instances. Additionally you can run commands with cook books against specific or all instances.

What to remember:

1. Stack has many layer
2. Each layer has instances
3. There are three different layers that following a rule set will come up or down
4. When the app comes up there will be some different events that will trigger different commands through cookbooks

OpsWorks is a lot of things: It is an instance manager, a layer manager, an app manager, does monitoring and all and all a integrated solution if you are using chef cookbooks.

Big thing to remember for the DevOps exam is the AWS OpsWorks Stacks LifeCycle Events

Each layer has a set of five lifecycle events, each of which has an associated step of recipes taht are specific ot the layer. When an event occurs on a layer’s instance, AWS OpsWorks Stacks automatically runs the appropriate set of recipes. We can the provide custom response to these events by implementing custom recipes and assigning them to the appropriate lifecycle events for each layer.

Lifecycle Events

Setup, Configure, Deploy, Undeploy, Shutdown.

Setup: this event occurs after a started instance has finished booting. You can also manually trigger the Setup event by using the Setup stack command. AWS OpsWorks Stacks runs recipes that set the instance up according to its layer.

Configure: This event occurs on all of the stack’s instances when one of the following occurs:

* An instance enters or leaves the online state
* You associate an Elastic IP address with an instance or disassociate one from an instance
* You attach an elastic Load Balancing load balancer to a layer or detach one from a layer.

This event is really important for configuring applications that need to know about all the other instances, for example in a distributed application.

Deploy: This event occurs when you run a Deploy command, typically to deploy an application to a set of application server instances. The instances run recipes that deploy the application and any related files from its repository to the layers instances.

Undeploy: this event occurs when you delta an app or run an Undeploy command to remove an app from a set of application server instances. The specified instances run recipes to remove all application versions and perform any required cleanup

Shutdown: This event occurs after you direct AWS OpsWorks Stacks to shut down an instance down but before the associated Amazon EC2 instance is actually terminated. AWS Opsworks Stacks runs recipes to perform cleanup tasks such as shutting down services.

If you have attached an Elastic Load Balancing load balancer to the layer and enabled support for connection draining, AWS OpsWorks stacks waits until conneciton draining is complete before triggering the Shutdown event.

After triggering a Shutdown event AWS OpsWorks Stacks allows Shutdown recipes a specified amount of time to perform their tasks, and then stops or terminates the Amazon EC2 instance. The default shutdown timeout is 120 seconds.

Key thing to remember here is that for the lifecycle stages of

Setup, Deploy, Undeploy, and Shutdown they are instance specific, unlike the lifecycle event of configure which will run on all instances when it is triggered.

OpsWorks and CloudWatch Events and Auto Healing

Every instance has an AWS OpsWorks Stacks agent that communicates regularly with the service. AWS OpsWorks Stacks uses that communication to monitor instance health. If an agent does not communicate with the service for more than approximately five minutes, AWS OpsWorks Stacks considers the instance to have failed.

In CloudWatch Events you can trigger an Event for the state change in the Instance

**CloudTrail:**

* <https://docs.aws.amazon.com/awscloudtrail/latest/userguide/cloudtrail-log-file-validation-cli.html>
* <https://docs.aws.amazon.com/awscloudtrail/latest/userguide/cloudtrail-receive-logs-from-multiple-accounts.html>
* <https://docs.aws.amazon.com/awscloudtrail/latest/userguide/cloudtrail-sharing-logs.html>

**CloudWatch:**

* <https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/cloudwatch_concepts.html#Metric>
* <https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/metrics-collected-by-CloudWatch-agent.html#linux-metrics-enabled-by-CloudWatch-agent>
* <https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/Counting404Responses.html>
* <https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/Subscriptions.html>
* <https://aws.amazon.com/blogs/big-data/power-data-ingestion-into-splunk-using-amazon-kinesis-data-firehose/>
* <https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/SubscriptionFilters.html#FirehoseExample>
* <https://docs.aws.amazon.com/AmazonCloudWatch/latest/events/Create-CloudWatch-Events-CloudTrail-Rule.html>
* <https://docs.aws.amazon.com/AmazonS3/latest/user-guide/enable-event-notifications.html>

**X-Ray**:

* <https://docs.aws.amazon.com/xray/latest/devguide/aws-xray.html>
* <https://aws.amazon.com/blogs/devops/using-amazon-cloudwatch-and-amazon-sns-to-notify-when-aws-x-ray-detects-elevated-levels-of-latency-errors-and-faults-in-your-application/>

**Amazon ES:**

* <https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/CWL_ES_Stream.html>

**Tagging in AWS**

* <https://aws.amazon.com/answers/account-management/aws-tagging-strategies/>

Cloud Trail

Track all api calls within the AWS account. However if you want them to be delivered to cloudwatch or s3 you need to create a cloud trail.

CloudTrail Logs are automatically encrypted however you can specify your own KMS key for encrypting as well. Since it can have a 15 minute time delay so not what you would want to use for immediate responses.

There are two outputs for cloudtrails you can output to S3 or a cloudwatch log group ( one line for each event )

You can validate a cloudtrail log file’s integrity using the AWS CLI

It will look for the modification of CloudTrail Log files, digest files

This is very useful for auditing your cloudtrails to identify if a attacker has removed the notification of the actions they took.

A digest file is only uploaded once every hour

CloudTrail Cross Account

You can have CloudTrail deliver log files from multiple AWS accounts into a single Amazon S3 bucket. For example you have four AWS accounts with different IDS. You then want to configure cloudtrail to deliver log files from all four of these accounts to a bucket belonging to one of the accounts.

1. Turn on cloudtrail in the account where the destination bucket will belong.
2. Update the bucket policy on your destination bucket to grant cross-account permissions to cloudtrail
3. Turn on cloudtrail in the other accounts you want. Configure CloudTrails in these accounts to use the same bucket belonging to the account that you originally set up the s3 for.

If you wanted to additionally allow for IAM users from other accounts to read the consolidated cloudtrail files you could create the policy that allows the users to only read parts of the s3 that are prefixed with their account number. Allowing for the access to their cloudtrail logs but preventing them from reading the other cloudtrail logs within the s3 bucket for other accounts.

AWS Kinesis Overview

Kinesis is a managed alternative to Apache Kafka

Great for applicaiton logs, metrics, IoT, clickstreams

Great for ‘real-time’ big data

Great for streaming processing frameworks like spark nifi etc

Data is automatically replicated to 3 AZ making it highly available

Kinesis Streams: low latency streaming ingest at scale

Kinesis Analytics: perform realtime analytics on streams using SQL

Kinesis Firehose: Load streams into S3, redshift, elastic search, etc.

Kinesis streams takes the input from all the different searches, then using kinesis analytics does any kind of metric or analytics on the information received and then finally utilizes kinesis firehose to then deposit the logs into a storage place. This is just one example and the three services can be utilized in different combinations as well, for example not doing kinesis analytics with the information coming in or directly sending all the information to kinesis firehose.

Kinesis Streams Overview

* Streams are divided in ordered shards / partitions
* Producers send in information which goes into the shards which are then read by consumers
* Data retention is 1 day by default and can go up to 7 days
* Ability to reprocess / replay data
* Multiple applications can consume the same stream
  + This is different from SQS which can only have one application consuming messages.
* Real time processing with scale of through put, the more shards you have the more throughput you will have
* Once data is inserted in kinesis it can’ be deleted ( immutability ) meaning the information will be in kinesis till after the data retention has been reached. This means you cannot delete data manually from kinesis.

KInesis Stream Shards

* One stream is made of many different shards
* Billing is per shard provisioned can have as many shards as you want.
* Batching available or per message calls
* The number of shards can evolve over time ( reshard / merge )
* Records are ordered per shard.
* So if you have a producer sending data and then consuming it. Each record is appended to each shard and will be ordered per shard but not across shards.

Kinesis Streams Records

* Data Blob: data being sent, serialized as bytes. Up to 1Mb. Cane represent anything
* Record Key
  + Sent alongside a record helps to group records in shards. Same key = Same shard.
  + Use a highly distributed key to avoid the “hot partition” problem
* Sequence Number: Unique identifier for each record put in shards. Added by kinesis after ingestion.

Kinesis Data Streams Limits to know

* Producer:
  + 1MB/s or 1000 messages/s at write PER SHARD
  + “ProvisionedThroughputException” is what is sent back if this is exceeded
* Consumer Classic:
  + 2MB/s at read PER SHARD across ALL consumers
  + 5 API calls per second PER SHARD across all consumers
  + = if 3 different applications are consuming, there is a possibility of throttling.
* Data Retention
  + 24 hours data retention by default
  + Can be extended to 7 days

Kinesis Producers

* Kinesis SDK
* Kinesis Producer Library
* Kinesis Agent
* CloudWatch Logs
* 3rd Party Libraries: Spark, Log4j, Appenders, Flume, Kafka Connect, NiFi etc.

Kinesis Consumers

* Kinesis SDK
* KInesis client Library
* Kinesis Connector Library
* Kinesis Firehose
* AWS Lambda
* 3rd Party Libraries: Spark, Log4j Appenders, Flume, Kafka, Connect…

There is additionally a special consumer unit to know which is the Kinesis KCL ( Kinesis Connector Library )

KCL uses DynamoDB to checkpoint offsets

KCL uses DynamoDB to track other workers and share the work amongst shards

Great for reading in a distributed manner.

Kinesis Data Firehose

* Fully Managed Service, no administration
* Near Real Time ( 60 seconds latency minimum for non full batches )
* Load data into Redshift / Amazon S3 / ElasticSearch / Splunk
* Automatic scaling
* Data Transformation through AWS Lambda ( ex: CSV to JSON )
* Supports compression when targe it Amazon S3 ( GZIP ZIP and SNAPPY )
* Pay for the amount of data going through firehose.

Things that can send information to kinesis data firehose

* SDK Kinesis Producer Library (KPL)
* KInesis Agent
* Kinesis Data Streams
* CloudWatch Logs & Events
* IoT rules actions

Lambda function to do transformations on the fly

Then outputs the data to s3, redshift, elastic search, or splunk

Kinesis Data Streams vs Firehose

* Streams
  + Going to write custom code ( Producer / Consumer )
  + Real Time ( ~200ms latency for classic )
  + Must manage scaling ( shard splitting and merging )
  + Data Storage for 1 to 7 days, replay capability for multiple consumers
  + Use with Lambda to inser data in real time to elastic search ( for example )
* Firehose
  + Fully managed send to s3, splunk, elasticsearch, redshift
  + Serverless data transformations with lambda
  + Near real time ( lowest buffer time is 1 minute )
  + Automatic Scaling
  + No data storage

AWS Kinesis data Analytics

* Perform real-time analytics on Kinesis Streams using SQL
* KInesis Data Analytics
  + Auto Scaling
  + Managed: no servers to provision
  + Continuous: real time
* Pay for actual consumption rate
* Can create streams out of the real time queries.

# 1/31/2023

AWS Cloudwatch in Depth

AWS Cloudwatch Metrics:

Multiple different services to view with specificity and generalization

Basic monitoring is included, but detailed monitoring costs more, the difference being how often metrics are taken. Detailed is in one minute, whereas basic is 5 minute intervals.

15 Months is the furthest information that is retained by cloudwatch

Metrics Retention

CloudWatch retains metric data as follows:

Data points witha period of less than 60 seconds are available for 3 hours.

Data points with a period of 60 seconds are available for 15 days

300 seconds are available for 63 days

3600 seconds 455 days / 15 months

Metrics to know

EC2

* CPU Utilization
* Disk Reads
  + Only instance store volumes
* Disk Read Operations
  + Only instance store volumes
* Disk Writes
  + Only instance store volumes
* Disc Write Operations
  + Only instance store volumes

If it has an EBS volume you can only get the read writes metrics from the EBS service

* Network In/Out
* Status Checks
* CPU Credit Usage and Balance
* Processes or RAM can be brought in throug cloudwatch and custom metrics and are not included in the base metrics

Elastic Block Store

* Read / Write Bandwith and Throughput
* Average Queue Length
* Idle Spent
* Read / Write Size
* How much space is left on the volume is again something that is not included by default and needs to be added through custom metrics.

Autoscaling

* Minimum Group Sizes
* Maximum Group Size
* Desired Capacity
* In Service Instances
* Pending Instances
* Standby Instances
* Terminating Instances
* Total Instances
* Group Metrics are NOT enabled for the auto scaling groups by default and can be enabled

Load Balancers

* Target Response Time
* Requests
* Rule Evaluations
* HTTP 5xxs
* HTTP 4xxs
* ELB 5xxs
* ELB 4xxs
* HTTP 500s, 501,502, 503, 504
* Target connection errors
* Sum rejected connections
* Target TLS negotiation errors
* Client TLS Negotiation error
* HTTP 3xxs
* HTTP 2xxs
* Active Connection Count
* New Connection Count
* Processed Bytes
* Consumed Load Balancer Capacity Units ( this is how AWS bills us for load balancers )

RDS

* CPU
* Database Connections
* Free Memory
* Read and Write Operations

In general you will get more metrics for managed services since AWS knows what to track and will setup more metrics.

Publishing Custom Metrics

* Standard Resolution, with data having a one minute granularity
* High Resolution, with data at a granularity of one second

Metrics produced by AWS services are standard resolution by default.

Dimensions:

In custom metrics, the –dimention parameter is common. A dimension further clarifies what the metric is and what data it stores. You can have up to 10 dimensions in one metric, and each dimension is defined by a name and value pair.

How you specify a dimension is different when you use different commands. With put-metric-data, you specify each dimension as MyName=MyValue, and with get-metric-statistics or put-metric-alarm you use the format Name Name=MyName and Value=MyValue.

You can either use the CLI, SDK or Cloudwatch Agent to push custom metrics.

There is no native function to export metrics out of cloudwatch.

Aws cloudwatch get-metrics-statistics –namespace value –metric-name name –dimensions here –statistics value –start-time starttime –end-time here –period period time – profile –region

Then a way to pipe this is to create a cloudwatch event that happens every hour that triggers a lambda that then pulls the logs from cloudwatch and saves them to s3

Cloudwatch Alarms are extremely important because they allow you to automate your infrastructure as a DevOps.

Based on your behavior or your services or applications and on the metrics that they publish

You can create a new alarm and it can be based on any metric including custom metrics. YOU CANNOT combine metrics for a single alarm they can only be based on a singular metric value and this can be a trick question on the exam.

After choosing a metric you then choose a condition for which the alarm to trigger.

Types:

* Static ( use a value as a threshold )
* Anomaly Detection ( use a band as a threshold )

Datapoints to Alarm, Defines the number of datapoints within the evaluation period that must be breaching the threshold to cause the alarm to go to the ALARM state.

Additionally you can trigger how the alarm behaves to missing data, for example it can be ignored or treated as a breach condition.

The Alarm Action

* SNS Topic
* Email

You can choose an action based on the alarms state a well

Autoscaling Action -> A way to autoscale up ECS or EC2

A thing to understand is that CloudWatch Events cannot be triggered from cloudwatch alarms. This is not a logical thing so it trips folks up. The way to get around this is to trigger a cloudwatch event from the SNS topic that the cloudwatch alarm publishes too.

Cloudwatch Billing Alarm

* Billing Alarm, create a new alarm on the cost of AWS services. This feature is only available in certain regions. ( eg. us-east-1 )

CloudWatch Logs

CloudWatch Unifined Agent for collecting metrics and logs from amazon EC2 instances and on premise servers with the cloudwatch agent.

The unified cloudwatch agent enabvles you do the following:

* Collect internal system-level metrics from EC2 instances across operating systems. The metrics includ in-guest metrics, in addition to the metrics for ec2 instances. Additional metrics that can be collected are listed in the docs
* Collect system level metrics from on-premise servers. These can include servers in a hybrid environment as well as servers not managed by AWS  
  Retrieve custom metrics from your paplicaitons oservices using ht eStatsD and collectd protocols Stats D is supported on both Linux servers and servers running Window Server. Collectd is supported on only linux servers
* Collect logs from amazon Ec2 instances sand on premise servers running either linudz or windows server

We need to additionally create an IAM role to puslogs from the cloudwatch agent

There is a managed policy

CloudWatchAgentServerPolicy

Cloud Watch Agent Shell Script

#Install the agent on amazon linux 2

Sudo yum install amazon-cloudwatch-agent

# Run the Wizard

Sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-config-wizard

/\* Run the wizard once and then store the configuration in SSM and then pull it for any subsequent settups instead of running the wizard. Additionally you will need to make sure the correct permissions for SSM are added to the role for it to do this properly. The difference here is that the CloudWatchAgentAdminPolicy has the proper permissions \*/

#Create some missing files

Sudo mkdir -p /usr/share/collectd

Sudo touch /user/share/collectd/types.db

#options

Sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a fetch-config -m ec2 -c ssm: AmazonCloudWatch-linud -s

Sudo /opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a fetch-confg -m ec2 -c file:/opt/aws/amazon-cloudwatch-agent/bin/config.json -s

Note this is how you could track the metrics for RAM usage for metrics.

Metric Filters:

This will take a log file and filter it based on a criteria

Create a filter pattern and a metric based on it and then additionally you can then create a cloudwatch alarm based on that new metric.

This is a way to create an alarm based on a filter of a log group.

You CAN export CloudWatch log data to S3 natively unlike CloudWatch Metrics

What happened if you wanted to analyze logs in real time and to do real time processing?

If we want to do this we will need to use subscriptions

Subscription is a way to get access to real-time feed of log events from CloudWatch Logs and have it delivered to other services such as Amazon Kinesis Stream, Amazon Kinesis Data Firehose, or AWS Lambda for custom processing, analytics, or loading to other systems.

To begin subscribing to log events create the receiving source, such as a Kinesis stream where the events will be delivered. A subscription filter defines the filter pattern to use for filtering which log events get delivered to your AWS resource, as well as information about where to send matching log events too.

All kinds of logs

* Application Logs
  + Logs that are produced by your application code
  + Contains cusom log messages, stack traces, and so on
  + Written to a local file system
  + Usually steamed to CloudWatch Logs using a CloudWatch Agent on EC2
  + If using Lambda, direct integration with cloudwatch logs
  + If using ECS or Fargate, direct integration with CloudWatch logs
  + If using Elastic Beanstalk, direct integration with Cloudwatch logs
* Operating System Logs ( Event Logs, System Logs )
  + Logs that are generated by your operating system ( EC2 or on-premise instance)
  + Informing you of system behavior ( ex: /var/log/messages or /var/log/auth.log
  + Usually streamed to CloudWatch Logs using a CloudWatch Agent
* Access Logs
  + List of all the requests for individual files that people have requested from a website
  + Example for httpd: /var/log/apache/access.log
  + Usually for load balancers, proxies, web servers, etc…
  + AWS provides some access logs
* Managed Logs
* Load Balancer Access Logs (ALB NLB CLB ) => to S3
  + Access logs for your load balancers
* CloudTrail Logs => to S3 and Cloudwatch logs
  + Logs for API calls made within your account
* VPC Flow Logs => to S3 and Cloudwatch logs
  + Information about IP traffic going to and from network interfaces in your VPC
* Route 53 Access Logs => to CloudWatch Logs
  + Log information about queries that ROute 53 receives
* S3 Access Logs => S3
  + Server access logging provides detailed records for the requests that are made to a bucket
* CloudFront Access Logs -> to S3
  + Detailed information about every user request that CloudFront receives

CloudWatch Events Overview

The bread and butter of devops. This is really how you bring all these services together in your account and build automations on top of them.

Event Bridge: -> Basically CloudWatch Events being rebranded with a little bit of stuff on it. Allowing you to have external events from external API providers onto cloud watch events but that is is really the only difference between the two.

Different options

Event Pattern or Schedule a CloudWatch Event

Either will trigger a target which can be multiple different things

There is a way to enable and configure event notifications for S3 buckets an they are completely unrelated to cloudwatch events.

In the S3 management console you can go to properties and then find the events section. Then define an event to listen for and then additionally add afilter for a prefix or suffix, then choose the target for the event to go to between lambda, sns or sqs

X Ray Overview

X ray is a service that collects data about requests tha your application serves and provides tools you can use to view filter and gain insights into that data to identify issues and opportunities for optimization. For any traced request to your application you can see detailed information not only about the request and response but also about calls that yourapplication makes to downstream AWS resources, microservices databases, and HTTP web APIs

X ray can be instrumented using the xray daemon or other SDK to directly send data into xray API.

What you get out ofit is throught he web browser, you’ll get a ageneral idea of the service mesh between your applications and how they relate to one another, the error rates and so on.

It’s able to drill down and allows you to dive deep and filter for traces and look at all the traces that were done within your accounts.

Amazon ElasticSearch

* May be called Amazon ES at the exam
* Managed version of ElasticSearch open source project
* Needs to run on servers ( not a serverless offering )
* Use cases:
  + Log Analytics
  + Real time application monitoring
  + Security Analytics
  + Full Text Search
  + Clickstream Analytics
  + Indexing

ELK stack

ElasticSearch + Kibana + Logstash

* ElasticSearch: provide search and indexing capability
  + You must specify instance types, multi-AZ, etc
* Kibana
  + Provid real time dashboards on top of the data that sits in ES
  + Alternative to CloudWatch dashboards ( more advanced capabilities )
* Logstash
  + Log ingestion mechanism, use the “Logstash Agent”
  + Alternative to CloudWatch Logs ( you can decide on log granularity and data retention )
  + An agent on an ec2 instance that send logs to elasticsearch

ElasticSearch Patterns DynamoDB

When we have a DYnamoDB table we see there is an api to retrieve and put items into it which is quite easy to use but only if we know the item that we are looking for in advance. If we have to find an item in the DDB without prior knowledge of it or the access pattern we need to do a scan action. However the scan action is really inefficient because we have to go through the entire dynamodb table. So it is quite common to ask for a searching mechanism on top of dynamo. DB and so through the integration of dynamo db streams that will send the data to a lambda function and that will then send the information to elasticsearch.

Then we could build an api on top of the elastic search to search for items, return for example the item IDs and then use this to retrieve the data itself from the dynamodb table.

Elastic Search Patterns CloudWatch Logs

To get cloudwatch logs into elastic search the first way we can use is the subscription filter. So we have cloud watch logs and we create a subscription filter for aws lambda which is real time.

Another way is to use a subscription filter to use it as a producer for a kinesis data firehose which then has a consumer of elastic search for the logs.

—--

Tagging in AWS

Tagging Strategies

AWS allows customers to assign metadata to their aws resources in the form of tags each tag is a simple label consisting of a key value pair that can make it easier to manage, search for and filter resources. Although there are no inherent types of tag, they enable customers to categorize resources by purpose, owner, environment, or any other criteria.

General Best Practices

When cheating a tagging strategy for AWS resources make sure that it accurately represents organizationally relevant dimensions and adheres to the following tagging best practices:

* Always use a standardized, case-sensitive format for tags, and implement it consistently across all resources types
* Consider tag dimensions that support the ability to manage resource access control, cost tracking, automation and organization
* Implement automated tools to help manage resource tags. The Resource Groups Tagging API enables programmatic control of tags making it easier to automatically manage, search and filter tags and resources. It also simplifies backups of tag data across all supported services with a single API call per AWS Region
* Err on the side of using too many tags rather than too few tags
* Remember that it iss easy to modify tags to accommodate changing business requirements, however consider the ramifications of future changes especially in relation to tag based access control, automation or upstream billing reports.

TBAC Tag Basic Access Control

Tagging Categories

* Technical Tags
  + Name
    - Used to identify individual resources
  + Applicaiton ID
    - Used identify disparate resources thate are related to a specific application
  + Application Role
    - Used to describe the function of a particular resource
  + Cluster
    - Used to identify resource farms that share a common configuration and perform a specific function for an application
  + Environment
    - Used to distinguis between development, test, and production infrastructure
  + Version
    - Used to help distinguis between different versions of resources or applications
* Tags for Automation
  + Date/Time
    - Used to identify the date or time a resources should be started, stopped, deleted, or routed
  + Opt in / Opt out
    - Used to indicate whether a resource should be automatically included in an automated activity such as starting , stopping, or resizing an instance
  + Security
    - Used to determine requirements such as encryption or enabling of VPC flow logs and also to identify route tables or security group s that deserve extra scrutiny
* Business Tags
  + Owner
    - Used to identify who is responsible for the resource
  + Cost Center / Business Unit
    - Used to identify the cost center or business unit associated with a resources typically for cost allocation and tracking
  + Customer
    - Used to identify a specific client that a particular group of resources serve
  + Project
    - Used to identify the project(s) the resource supports
* Security Tags
  + Confidentiality
    - An identifier for specific data confidentiality levels a resource supports
  + Compliance
    - An identifier for workloads designed to adhere to specific compliance requirements.

**AWS Systems Manager**

* <https://docs.aws.amazon.com/systems-manager/latest/userguide/systems-manager-managedinstances.html>
* <https://docs.aws.amazon.com/systems-manager/latest/userguide/activations.html>
* <https://docs.aws.amazon.com/systems-manager/latest/userguide/patch-manager-approved-rejected-package-name-formats.html>
* <https://docs.aws.amazon.com/systems-manager/latest/userguide/systems-manager-automation.html>
* <https://docs.aws.amazon.com/systems-manager/latest/userguide/automation-walk-patch-linux-ami-console.html>
* <https://docs.aws.amazon.com/systems-manager/latest/userguide/automation-cf.html>
* <https://d1.awsstatic.com/whitepapers/aws-building-ami-factory-process-using-ec2-ssm-marketplace-and-service-catalog.pdf>
* <https://github.com/miztiik/AWS-Demos/tree/master/How-To/setup-ami-lifecycle-management-using-ssm>

**AWS Config**

* <https://docs.aws.amazon.com/config/latest/developerguide/s3-bucket-policy.html>
* <https://aws.amazon.com/about-aws/whats-new/2019/03/use-aws-config-to-remediate-noncompliant-resources/>)
* <https://aws.amazon.com/about-aws/whats-new/2018/03/aws-config-notifications-are-now-integrated-with-amazon-cloudwatch-events/>
* <https://docs.aws.amazon.com/config/latest/developerguide/monitor-config-with-cloudwatchevents.html>
* <https://docs.aws.amazon.com/config/latest/developerguide/notifications-for-AWS-Config.html>
* <https://docs.aws.amazon.com/config/latest/developerguide/aggregate-data.html>

**AWS Inspector**

* <https://aws.amazon.com/about-aws/whats-new/2017/07/amazon-inspector-adds-event-triggers-to-automatically-run-assessments/>
* <https://aws.amazon.com/blogs/security/how-to-remediate-amazon-inspector-security-findings-automatically/>
* <https://aws.amazon.com/blogs/security/how-to-set-up-continuous-golden-ami-vulnerability-assessments-with-amazon-inspector/>
* <https://d1.awsstatic.com/whitepapers/aws-building-ami-factory-process-using-ec2-ssm-marketplace-and-service-catalog.pdf>
* <https://aws.amazon.com/blogs/awsmarketplace/announcing-the-golden-ami-pipeline>

**AWS Health**

* <https://status.aws.amazon.com/>
* <https://github.com/aws/aws-health-tools/tree/master/automated-actions/AWS_RISK_CREDENTIALS_EXPOSED>

**AWS Trusted Advisor**

* <https://docs.aws.amazon.com/awssupport/latest/user/cloudwatch-events-ta.html>
* <https://github.com/aws/Trusted-Advisor-Tools>
* <https://github.com/aws/Trusted-Advisor-Tools/tree/master/ExposedAccessKeys>
* <https://github.com/aws/Trusted-Advisor-Tools/tree/master/HighUtilizationEC2Instances>
* <https://docs.aws.amazon.com/awssupport/latest/user/cloudwatch-metrics-ta.html>
* <https://aws.amazon.com/premiumsupport/technology/trusted-advisor/>
* <https://docs.aws.amazon.com/cli/latest/reference/support/index.html#cli-aws-support>

**AWS Macie**

* <https://aws.amazon.com/blogs/security/classify-sensitive-data-in-your-environment-using-amazon-macie/>

AWS System Manager OVerview

°Helps you manage your EC2 and On-Premise systems at scale

Get operational insights about the state of your infrastructure

Easily detec problems

Patching automation for enhanced compliance

Works for both windows and linux os

Integrated with cloudwatch metrics and dashboards

Integrated with aws config

Free service

Resource groups

Insights

* Insights dashboard
* Inventory discover and audit the softwar installed
* Compliance

Parameter store

Aciton:

* Automation shutdown ec2 and create ami
* Run commands
* Session manager
* Patch manager
* Maintenance windows
* State Manager define and maintaining configuration of OS and applications

HOw Systems Manager Works

* We need to install the SSM agent onto the systems we control
* INstalled by default on Amazon Linux amis and some Ubunto AMIs
* If an instance can’t be controlled with SSM it’s probably an issue with the SSM agent!
* Make sure the EC2 instances have a proper IAM role to allow ssm actions

System Manager

Activating on premise systems with Systems Manager:

#Create an directory for the ssm agent

Mkdir /tmp/ssm

Curl <https://s3.amazonaws.com/ec2-downloads-windows/SSMAgent/latest/linux_amd64/amazon-ssm-agent.rpm> -o /tmp/ssm/amazon-ssm-agent.rpm

Sudo yum install -y /tmp/ssm/amazon-ssm-agent.rpm

Sudo systmctl stop amazon-ssm-agent

#edit the code, id and region in the command below

Sudo amazon-ssm-agent -register -code “activation-code” -id “activation-id” -region “region

Sudo systemctl start amazon-ssm-agent

Hybrid activations, create activation code, IAM role creation for the instance, upon creation you will get an activation code and an activation id as well.

On premise instances in ssm are prefixed with mi- whereas the instances that come with the SSM agent are prefixed with i-

Resource Groups, way to group resources together

Tag based, or Cloudformation Stack based

Then you can choose the tag or cloudformation as well to group the resources attached to a tag

SSM Run Command

SSM Documents

Documents are JSON files that can either be owned by AWS or Yourself

Amongst them there are four types of Documents

1. Command
2. Automation
   1. Automation will be used when we do automation tasks
3. Policy
4. Session

Good examples

AWS UpdateSSMAgent which will run a command on all managed instances to update the ssm agent to the most recent version

AWS RunShellScript We run a shell script on all managed instances

Now you can choose the target that you want to run the command against in this case there are three different options. You can specify instance tags, choose instances manually or choose a SSM resource group.

Rate Control, how to run the command

Concurrency or Error threshold

Output Options => Dump logs to either s3 or cloudwatch

This is a way to run one off commands on different instances instead of doing it manually on the different instances.

SSM Parameter Store

Custom naming system for the parameter store, however there is usually a naming practice for parameters based on the company

There are a few different opinions including, string, string list, and secure string ( a way to store secret strings )

Additionally there is a versioning and audit trail for changes to a parameter in the parameter store.

#GET PARAMETERS

Aws ssm get-parameters –names parameterNameHere additionalParameterNameHere

#GET PARAMETERS WITH DECRYPTION

Aws ssm get-parameters –names parameterNameHere additionalParameterNameHere –with-decryption

#GET PARAMETERS BY PATH  
Aws ssm get-parameters-by-path –path pathPatternHere

#GET PARAMETERS BY PATH RECURSIVE

Aws ssm get-parameters-by-path –path pathPatternHere –recursive

#GET PARAMETERS BY PATH RECURSIVE WITH DECRYPTION

Aws ssm get-parameters-by-path –path pathPatternHere –recursive –with-decryption

SSM Patch Manager

There are a few different options including the configure patching, and additionally predefined baselines. There are a bunch of baselines defined in accounts that tell the patch manager how to patch by default

Create a patch baseline

Name, OS, and if the patch is a baseline. ( all instances that are that type will be patched to this new baseline )

You can then specify approved and rejecte patches from a list of patch file names

Maintence windows will provide a window of time for commands to be run

SSM Inventory, once set up will give you information on what is installed on a given instance

Automation

Ways to do Automated steps in order using different documents

AWS Config:

Provides an inventory of your AWS resources and a history of configuration changes to these resources. You can use AWS Config to definerules that evaluate these configurations for compliance

AWS Service Catalog

Users that are new to AWS have too many options and may create stacks that are not compliant / in line with the rest of the organization

Some users just want a quick self service portal to launch a set of authorized products predefined by admins

Includes virtual machines databases storage options etc.

Create and manage catalogs of IT services that are approved on AWS

The products are cloudformation templates

Ex virtual machines images, servers, software, databases, regions, ip address ranges

Cloudformation helps ensure consistency and standardization by admins

They are assigned to portfolios ( teams )

Teams are presented a self service portal where they can launch the products

All the deployed products are centrally managed deployed services

Helps with governance, compliance, and consistency

Can give user access to launching products without requiring a deep knowledge of AWS

Integrations with self service portals such as service now.

Amazon Inspector

Amazon Inspector is an automated vulnerability management service that continually scans AWS workloads for software vulnerabilities and unintended network exposure.

EC2 Instance Compliance

AWS Config

* Ensure instance has proper AWS configuration ( not open SSH port, etc )
* Audit and compliance over time

Inspector

* Security Vulnerabilities scan from within the OS using the agent
* Or outside network scanning ( no need for the agent )

Systems Manager

* Run automations, patches, commands, inventory at scale

Service Catalog

* Restrict how the EC2 instances can be launched to minimize configurations
* Helpful to onboard beginners to AWS

Configuration Management

* SSM, OpsWorks, Ansible, Chef, Puppet, User Data
* Ensure EC2 instances have proper configuration files

AWS Service Health Dashboard

Status.aws.amazon.com

This provides a global view of the health of a given service in a given region within aws

To get a more personalized health board you can use the Personal Health Board

In this dashboard you can see what the issues are and what of your resources may have been affected etc.

Additionally you can trigger cloudwatch events with the personal health board to help notify if something has been effected within your environment.

Trusted Advisor

In automating the limits notifications in case you do reach them and automating some security events.

Trusted Advisor is going to have recommendations for your accounts on five categories.

Cost Optimization, Performance, Security, Fault Tolerance, and Service Limits

There is a free and paid tier, and certain recommendations are only enabled in the paid tear. For example the cost optimization and performance are good examples of which ones we need the paid tier for.

Some of the security and service limits are the ones that are included in the free tier.

Automations with trusted advisor, one of the ways to do this is to integrate with Cloudwatch Events, example triggering a cloudwatch event to send a slack message when a service limit is almost met.

AWS Trusted Advisor only refreshes when visiting the page or clicking on the refresh button. There is a way to use the AWS CLI to refresh the trusted advisor throught the support api.

AWS GuardDuty

Intellegen Threat discovery to protect AWS account

Uses machine learning algo anomaly detection 3rd party data

One click to enable no need to install software

Input data includes:

* Cloudtrail logs: unusual api calls, unauthorized deployments
* VPC flow logs: unusual internal traffic patterns, unusual ip addresses
* DNS Logs comprised of ec2 instance s sending encoded data within DNS queries

Notifies you in case of findings

Integration with aws lambda

Amazon Macie:

A data visibility security service that helps classify and protect your sensitive and business critical content.

Secrets Manager:

A service to store secrets very much like the parameter stores secure string the big difference is that you can rotate the credentials easily. There is additionally tight coupling features with database services on aws

AWS License Manager:

A way to manage licenses from third party vendor etc.

AWS Cost Allocation Tags:

* With Tags we can track resources that relate to each other
* With Cost Allocation Tags we can enable detailed cost reports
* Just like Tags, but they show up as columns in reports
* AWS Generated Cost Allocation Tags
  + Automatically applied ot the resources you create
  + Stars with Prefix aws
  + They’re not applied to resources created before the activation
* User tags
  + Defined by the user
  + Starts with the prefix of user
* Cost Allocation Tags just appear in the billing console
* Takes up to 24 hours for the tags to show up in the report.

AWS Data Protection

* TLS for in transit encryption
* ACM to manage SSL / TLS certificates
* Load Balancers
  + ELB, ALB & NLB profvide SSL termination
  + Possible to have multiple SL certificates per ALB
  + Optional SSL/TLS encryption between ALB and EC2 instances ( else HTTP )
* Cloudfront with SSL
* ALl AWS services expose HTTPS endpoints
* You could but shouldnt use HTTP with S3

AWS Data Protection At Rest Encryption

S3 Encryption

* SSE-S3 Serverside Encryption using AWS’s key
* SSE-KMS server side encryption using you own KMS key
* SSE-C Server side encryptionby providing your own key AWS won’t keep it
* Client side encryption send encrypted content to aws no knowledge of the key whatsoever
* Possibility to enable default encryption on se throught he settings
* Possibilit to enforce through s3 bucket policy
* Glacier is encrypted by default

One quick setting for EBS EFS RDS Elasticache DynamoDB etc

* Usually uses either service encryption key or your own KMS key

Category of Data

* PHI = Protected Health information
* PII = Personally identifying information

AWS Network Proteciton

* Direct Connect: private, direct connection between site and AWS
* Public Internet: use a VPN
  + Site to site bpn supports internet protocol security IPSEC VPN connections
* Network ACL: stateless firewall at the VPC level
* WAF ( Web application firewall) web security rules against exploits
* Security Groups stateful firewall on the instance underlying hypervisor
  + If traffic is allowed in it will be allowed out
* System Firewalls: install your own firewall on EC2 instances.

**Auto Scaling**

* <https://gist.github.com/mikepfeiffer/d27f5c478bef92e8aff4241154b77e54>
* <https://aws.amazon.com/about-aws/whats-new/2018/05/application-load-balancer-announces-slow-start-support/>
* <http://docs.aws.amazon.com/autoscaling/latest/userguide/as-suspend-resume-processes.html>
* <https://docs.aws.amazon.com/autoscaling/ec2/userguide/lifecycle-hooks.html>
* <https://github.com/aws-samples/aws-lambda-lifecycle-hooks-function>
* <https://docs.aws.amazon.com/autoscaling/ec2/userguide/as-instance-termination.html>
* <https://aws.amazon.com/about-aws/whats-new/2015/12/protect-instances-from-termination-by-auto-scaling/>
* <https://aws.amazon.com/blogs/devops/use-a-creationpolicy-to-wait-for-on-instance-configurations/>
* <https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-attribute-updatepolicy.html>
* <https://aws.amazon.com/premiumsupport/knowledge-center/auto-scaling-group-rolling-updates/>
* <https://docs.aws.amazon.com/codedeploy/latest/userguide/tutorials-auto-scaling-group-create-deployment.html>
* <https://docs.aws.amazon.com/codedeploy/latest/userguide/integrations-aws-auto-scaling.html>
* <https://d1.awsstatic.com/whitepapers/AWS_Blue_Green_Deployments.pdf>

**DynamoDB**

* <https://aws.amazon.com/blogs/database/choosing-the-right-dynamodb-partition-key/>
* <https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/AppendixSampleTables.html#AppendixSampleData.Thread>
* <https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Streams.html#Streams.Processing>
* <https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/time-to-live-ttl-how-to.html>
* <https://aws.amazon.com/blogs/big-data/building-and-maintaining-an-amazon-s3-metadata-index-without-servers/>

**Multi-AZ**

* <https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/EBSEncryption.html>

**Multi-Region**

* <https://docs.aws.amazon.com/config/latest/developerguide/aggregate-data.html>
* <https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/stacksets-getting-started.html>
* <https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/stacksets-prereqs.html>
* <https://docs.aws.amazon.com/codepipeline/latest/userguide/actions-create-cross-region.html>
* <https://aws.amazon.com/blogs/devops/using-aws-codepipeline-to-perform-multi-region-deployments/>

**Disaster Recovery**

* <https://docs.aws.amazon.com/efs/latest/ug/efs-backup-solutions.html>
* <https://docs.aws.amazon.com/Route53/latest/DeveloperGuide/hosted-zones-migrating.html>
* <https://stackoverflow.com/questions/20337749/exporting-dns-zonefile-from-amazon-route-53>
* <https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/environment-configuration-methods-before.html#configuration-options-before-savedconfig>
* <https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/environment-configuration-savedconfig.html>

**On-Premise**

* <https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/amazon-linux-2-virtual-machine.html>
* <https://aws.amazon.com/application-discovery/>

**Multi-Account**

* <https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/CrossAccountSubscriptions.html>
* <https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/CreateDestination.html>
* <https://aws.amazon.com/premiumsupport/knowledge-center/streaming-cloudwatch-logs/>
* <https://aws.amazon.com/blogs/architecture/stream-amazon-cloudwatch-logs-to-a-centralized-account-for-audit-and-analysis/>
* <https://aws.amazon.com/blogs/architecture/stream-amazon-cloudwatch-logs-to-a-centralized-account-for-audit-and-analysis/>
* <https://aws.amazon.com/premiumsupport/knowledge-center/streaming-cloudwatch-logs/>
* <https://aws.amazon.com/about-aws/whats-new/2017/06/cloudwatch-events-adds-cross-account-event-delivery-support/>
* <https://docs.aws.amazon.com/AmazonCloudWatch/latest/events/CloudWatchEvents-CrossAccountEventDelivery.html>

# 2/1/2023

Domain 5 Coverage Overview

Troubleshoot issues and determine how to restore operations

* CloudWatch, CloudFormation, Rollbacks, Etc.

Determine how to automate event management and alerting + apply concepts required to setup event driven automated actions

* CloudWatch Events, CloudWatch Alarms, SNS

Automated Healing

* Cloudformation ( triggered by an alarm )
* Beanstalk (easier)
* OpsWorks (automatic host replacement manages the infrastructure)
* AutoScaling( Is the big importance of domain 6 )

ASG Auto Scaling Groups

From Launch Configuration: Setting up what the ami you will be using any additional commands to be run on setup etc. From here when the auto scaling group goes up and down it will spin up instances based on the launch configuration provided.

Launch Templates

Use launch templates to automate instance launches, simplify permission policies, and enforce best practices across your organization. Save launch parameters in a template that can be used on demand launches and with managed services including EC2 auto scaling and ec2 fleet. This will allow you to update launch parameters by creating a new version of the launch template.

Scheduled Actions

With these actions we can change the number of instances and when the recurrence should happen and when the start and end time for the action.

Scaling Policies

Default Cooldown, is the number of seconds that will pass before another scaling activity can occur. So if you set it too low, you may have instances coming in and out too quickly where if you set it too high the application may not scale fast enough for the amount of traffic.

A scaling policy will look at a specific metric and then the target value for that metric and determine how many instances need to scale in for the policy

An important thing to note here is that the instance needs x seconds to warm up after scaling and if this value is set higher than the default cool down additional scaling will occur because the new instance is not yet factored into the reason for why we are scaling our instances in the first place.

Three types of scaling polices you have target, simple, and step

HTTP to HTTPS

Suspending and resuming a process for an auto scaling group

Types of processes

* Launch
  + Adds instances ot the Auto Scaling group when the group scales out, or when Amazon EC2 Auto scaling chooses to launch instances for other reasons such as when it adds instances to a warm pool
* Terminate
  + Removes instances from the ASG, or when the ASG chooses to terminate instances for other reasons. IE: maximum lifetime duration or failing health check
* AddToLoadBalancer
  + Adds the instances to teh attached load balancer
* AlarmNotification
  + Accepts notifications from cloudwatch alarms taht are associated with dynamic scaling policies
* AZRebalance
  + Making sure instances are spread evenly across AZs
* HealthCheck
  + Checks instance health
* InstanceRefresh
  + Terminates and replaces instances using the instance refresh feature
* ReplaceUnhealthy
  + Terminates instances that are marked as unhealthy and then creates new instances to replace them with.
* ScheduledActions
  + Performs scheduled scaling actions

Considerations

Before suspending process consider the following

* You can suspend and resume individual processes or all processes
* Suspending a process affects all instances within the ASG
* Suspending AlarmNotification allow syou to temporarily stop the groups target tracking, step, and simple scaling policies without deleting the scaling policies or their associated cloudwatch alarms.
* If you suspend the Launch or Terminate processes, or AZRebalance, and then you make changes to the ASG your group can become unbalanced between AZs If that happens, upon resuming processes the ASG will gradually redistrbutes instances evenly between the AZs
* Suspending the Terminate processes doesn’t prevent the successful termination of instances using the force delete option with the delete-auto-scaling-group command

Understanding how suspending processes affects other processes

1. Launch is suspended
   1. AlarmNotificaiton is still active, but the ASG can’t initiate scale-out activities for alarms that are in breach
   2. ScheduledActions is active, but ASG can’t initiate scale out activities for any of the scheduled actions
   3. AZRebalancing can’t rebalance across AZs
   4. ReplaceUnhealthy continues to terminate unhealthy instances however it will not be able to replace the instances.
   5. InstanceRefresh will not replace instances
2. Terminate is suspended
   1. AlarmNotification is still active, but the ASG can’t initiate scale in activities for alarms
   2. ScheduledActions are active but the ASG can’t initiate scale in actions
   3. AZRebalance, it will continue to launch new instances without terminating the old ones. This can cause the ASG to grow up to 10% than its max size
   4. ReplaceUnhealthy is inactive but not HealthCheck, so it will mark unhealthy instances but does not replace them
   5. InstanceRefresh will not replace instances

More Here <https://docs.google.com/document/d/1t6YhGblFZ2-Gb1B1BbAT_hK_v31o5MxtoWlcR8gKM1Q/edit>

Detach / Stand-By Mode / Scale In Protection

Detaching an instance that is in the InService state from an ASG it will remove it from the ASG and the asg will not interact with it anymore

StandBy, a state for which you can then update or troubleshoot the particular instance then return it to service without having to detach it from the ASG. When an instance is on standby they are still part of the auto scaling group but they do not actively handle load balancer traffic. However when put into standby the ASG can become unbalanced between AZs. Auto Scaling compenstates by rebalancing the AZs unless the AZRebalance process was additionally suspended

How it works

1. The instance remains in the standby state till put back into service
2. The load balancer deregisters the instance. There is a 300 second default for letting the instance drain before deregistering it
3. By default the specified desired capacity is decremented when placing an instance on standby this prevents the additional launch of other instances.
4. You can update or troubleshoot the instance now
5. Return the instance back to service
6. The desired capacity is then incremented
7. If there is a load balancer it will reregister the instance

Scale In protection

How this one works is that it prevents the termination of the instance in question from being scaled in.

AutoScaling Lifecycle Hook

Three ways to add notifications within the lifecycle hook

SNS, SQS, CloudWatch ( which is now the recommended way )

Instance Launch and Instance Terminate

Then from these an api call has to be made to tell the instance in the ASG to continue in the lifecycle ( there is a timeout as well for it move forward as well )

A common example question is how to have the EC2 run a command on the lifecycle hook. The answer is that upon triggering the lifecycle hook it reaches the lambda which will then trigger an SSM Run Command for that particular EC2 instance

ASG

Termination Policies

Default Policy

1. Determine which AZs have the most instance, and at least one instance sthat is not protected from scale in. If there are multiple unprotected instances to choose from in an AZ it chooses an instance based on the following criteria
   1. [for ASG with multiple instance types and purchase only option ]Determine which instance to terminate as to align the remaining instances to the allocation strategy for the on demand or spot instance that is terminating and you current selection of instance types. If there is one such instance terminate it otherwise apply the next condition
   2. [For ASG that use a launch template] determine whether any of the instances use the oldest launch template. If there is one such instance terminate it. ( There is one exception: if the group originally used a launch configuratio. ASG terminates instances that use launch configuration before instances with launch templates. )
   3. [ ASG tha use a launch configuration ] Determine wheter any of the instances use the oldest launch configuration. If there is one such instance terminate it
2. After applyin all of the criteria a boce if ther are multiple unprotected instances to terminate determine which instances are closest to the next billing hour. If there is one such instance terminat it. ( Terminating the instance closest to the next billing hour helps you maximize the use of your instances that have an hourly charge )
3. If there still is not an instance identified it will choose one of the instances at random from the group of instances.

Custom Termination Policy Options

* OldestInstance
* Newest Instance
* OldestLaunchConfiguration
* ClosestToNextInstanceHour
* Default
* OldestLaunchtemplate
* AllocationStrategy

With ASG cloudformation the importan note is that when updating a ASG template you need to add a UpdatePolicy in the cloudformation or else only the ASG will be updated but the instances will not update.

If codedeploy is integrated with ASG it can do some very useful things.

CodeDeploy can deploy revisions to the new instances automatically. You can also coordinate deployments in codedeploy with AS instances when they are no long er needed.

When new instances are launched as part of a ASG code deploy can deploy the revisions to the new instances automatically. It can also coordinate deployments with instances registered with elastic load balancing as well

In order for codedeploy to deploy application revisions to new ec2 instances during a scale out event code deploy uses an ASG lifecycle hook. The lifecycle hook notifies code deploy that an scale out event is in progress and that codedeploy must deploy a revision to the scaled out instance

<https://docs.aws.amazon.com/codedeploy/latest/userguide/integrations-aws-auto-scaling.html>

Deployment strategies

Auto Scaling and ALB

In Place ( one LB, one TG, one ASG )

Rolling ( one LB, one TG, one ASG new instances )

Replace ( one LB, one TG, two ASG new instances )

Blue / Green ( Two LB, Two TG, Two ASG, new instances and R53 )

Read the blue green deployments on AWS whitepaper

DynamoDB

Local Secondary Indexes cannot be created after the table has been provisioned.

LSI the PK is the same the sort key can be different.

DAX

This is a dynamodb accelerator for caching information. This needs to be created in advance for the DDB table

An important note for dynamodb streams is that because the underlying service is kinesis you can’t have more than three consumers. This means that if you want three lambdas to respond to a change in your DB you need to have an intermediary SNS topic that allows for more consumers to respond to the change. This is caused by hot sharding problems.

DDB Patterns S3 metadata index

Write a lot of objects to s3, and we want lambda responds and writes the metadata to a dynamo db. This way we can create an api around the meta data.

* Search by date
* Total storage used by a customer
* List of all objects with certain attributes
* Find all objects uploaded within a date range

DDB and Elastic Search

An api to retrieve items go directly to Dynamo other wise all information is written to Elastic search via streams and lambda functions. Then to search for items you use an api that interacts with ElasticSearch. This is because scan functions in DDB are incredibly resource heavy.

Multi AZ in AWS

* Services where multi az must be enabled manually
  + EFS, ELB, ASG, Beanstalk: assigned AZ
  + RDS, ElastiCache: multi AZ ( synchronous standby DB for failovers )
  + Aurora
    - Data is stored automatically across multi az
    - Can have multi-az for the db itself
  + ElastiSearch ( managed) multi master
  + Jenkins (self deployed) multi master
* Services where multi az is implicitly there
  + S3 (except for the onezone infrequent access tier)
  + DynamoDB
  + All of AWS proprietary managed services

EBS is tied to a single AZ

How can you make EBS “multi az”

* ASG with min max desired
* Lifecycle hooks for terminate make a snapshot for the EBS volume
* Lifecycle hook for start: copy the snapshot create an EBS attach to instance.

Note: for PIOPS volume to get max performance after snapshot read the entire volume once allowing for the pre warming of the IO blocks

Multi Region Services

DynamoDB Global Tables (Multi way replication enabled by streams )

AWS config aggregators ( multi region and multi account )

RDS Cross Region Read Replicas ( used for Read and DR)

Aurora Global Database ( one region is master other is for read & DR )

EBS volumes snapshots, AMI, RDS snapshots can be copied across regions

VPC peering to allow private traffic between regions

Route53 uses a global network of DNS servers

S3 Cross Region Replication

CloudFront for Global CDN at the edge locations

Lambda@Edge for Global Lambda function at edge locations ( A/B testing )

Multi Region with Route 53

* Health Check => automated DNS failovers
  + Health checks that monitor an endpoint ( application server or other aws resources )
  + Health checks that monitor other health checks ( calculated health checs
  + Health checks that monitor CloudWatch alarms (full control ) eg throttles of a DDB custom metrics etc
  + Health Checks are integrated with CloudWatch Metrics

CloudFormation StackSets

* Create update or delete stacks across multiple accounts and regions with a single operation
* Administrator account to create stacksets
* Trusted accounts to create update delete stack instances from stack sets
* When you update a stack set, all associated stack instances are updated throughout all accounts and regions
* Ability to set a maximum concurrent actions on targets ( # or % )
* Ability to set failure tolerance ( # or % )

Code Pipeline for multi region deployments.

Disaster Recovery Overview

* Any event that has a negative impact on a company’s business continuity or finances is a disaster
* Disaster Recovery (DR) is about preparing for and recovering from a disaster
* What kind of disaster recovery?
  + On-Premise => On-Premise: traditional DR
  + On-Premise => AWS Cloud: Hybrid recovery
  + AWS Cloud Region A => AWS Cloud Region B
* Need to define two terms
  + RPO: Recovery Point Objective
  + RTO: Recovery Time Objective

RPO and RTO

RPO is the point in time at which you go back and defines how much time are you ok with losing.

RTO is the amount of down time you experience before becoming operational again

Disaster Recovery Strategies

* Backup and Restore
* Pilot Light
* Warm Standby
* Hot Site / Multi Site Approach

Backup and Restore ( High RPO )

This means you backup information and the down time is spinning things up again from the backup. This is a very easy DR, and fairly inexpensive, however you have a high RPO.

Disaster Recovery- Pilot Light

* A small version of the app is always running in the cloud
* Useful for the critical core ( pilot light )
* Very similar to backup and restore
* Faster than backup and restore as critical systems are already up

Warm Standby

* Full system is up and running, but at a minimum size
* Upon disaster we can scale to production load

Multi / Hot Site Approach

* Very low RTO - but very expensive
* Full Production Scale is running on AWS and On Premise

All AWS Region

* Two production scales running in two different AWS Regions

Disaster Recovery Tips

* Backup
  + EBS Snapshots, RDS automated backups, Snapshots, etc
  + Regular pushes to S3 / S3 IA / Glacier, Lifecycle Policy, Cross Region Replication
  + From on Premise: snowball or storage gateway
* High Availability
  + Use route53 to migrate DNS over from Region to Region
  + RDS Multi-AZ, ElastiCache Multi-AZ, EFS, S3
  + Site to Site VPN as a recovery from Direct Connect
* Replication
  + RDS Replication ( Cross Region ), AWS Aurora + Global Databases
  + Database Replication from on-premise to RDS
  + Storage Gateway
* Automation
  + CloudFormation / ElasticBeanstalk to recreate a whole new environment
  + Recover / Reboot EC2 instances with CloudWatch if alarms fail
  + AWS Lambda functions for customized automations

Multi-Region Disaster Recovery Checklist

* Is my AMI copied? Is it stored in the parameter store
* Is my cloudFormation StackSet working and tested to work in another region
* What’s my RPO and RTO
* Are Route53 Helath Checks working correctly? Tied to a CW Alarm?
* How can I automate with CloudWatch Events to Trigger some Lambda functions and perform a RDS Read Rpelicaiton promotion
* Is my data backed up? EBS, AMI, RDS, S3, CRR, Global DynamoDB tables, RDS, RDS & Aurora Global Read Replicas

Backups & Multi-Region DR

* EFS Backup
  + AWS Backup with EFS ( frequency, when, retain, time, lifecycle policy ) - managed
  + EFS to EFS backup ( most likely obsolete in favor of AWS Backup )
  + Multi-region idea: EFS => S3 => S3 CRR => EFS
* Route 53 Backup
  + Use ListResourceRecordSet API for exports
  + Write your own script for imports into R53 or other DNS provider
* Elastic Beanstalk Backup
  + Saved configuration using the eb cli or aws console

On-Premise strategy with AWS

* Ability to download Amazon Linux 2 AMI as a VM in (.iso format)
  + VMWare, KVM, VirtualBox, Microsoft Hyper V
* VM Import / Export
  + Migrate existing applications into EC2
  + Create a DR repository strategy for you on premise VMs
  + Can export back the VMs from EC2 to on Premise
* AWS Application Discovery Service
  + Gather information about your on premise servers to plan a migration
  + Server utilization and dependency mappings
  + Track with AWS Migraiton Hub
* AWS Database Migration Service ( DMS )
  + Replicate on premise => AWS, AWS=> AWS, AWS => On premise
  + Works with various database technologies
* AWS Server migration service
  + Incremental replication of on-premise live servers to AWS

AWS Organizations

* Global Service
* Allows to manage multiple AWS accounts
* The main account is the master account - you can’t change it
* Other accounts are member accounts
* Member accounts can only be part of one organization
* Consolidaited Billing across all accounts - single payment method
* Pricing benefits from aggregated usage ( EG volume discount for EC2, S3….)
* API is available to automate AWS account creation

Multi Account Strategies

* Create accounts per department, per cost center, per envirionment, based on regulatory restrictions ( using SCP ) for better resource isolation, to have separate per account service limits isolated account for logging
* Multi Account vs One Account Multi VPC
* Use tagging standards for billing purposes
* Enable CloudTrail on all accounts, send logs to central S3 account
* Send CloudWatch Logs to central logging account
* Establish Cross Account Roles for Admin Purposes

Organizational Units is the different uses where you define multiple children aws accounts under the root ou master account

Service Control Polices (SCP)

* Whitelist or backlist IAM actions
* Applied at the OU or Account level
* Does not apply to the master account
* SCP is appleid to all the users and roles of the account including root
* The SCP does not affect service linked roles
  + Service linked roles enable other aws services to integrate with AWS organizations and can’t be restricted by SCPs
* SCP must have an explicit allow does not allow anything by default
* Use cases:
  + Restrict access to certain services for example can’t use EMR
  + Enforce PCI compliance by explicitly disabling services

AWS Organization Moving Accounts

To migrate accounts from one organization to another

1. Remove the member account from the old organization
2. Send an invite to a the new organization
3. Accept the invite to the new organization from the member account

If you want the master account of the old organization to also join the new organization do the following

1. Remove the member accounts from the organizations using the procedure above
2. Delete the old organization
3. Repeat the process above to invite the old master account to the new organization

Multi Account with AWS

* Any cross account action requires ot define IAM “trust”
* IAM roles can assumed across account
  + No need to share IAM creds
  + Uses AWS Security Token Service (STS)
* Code Pipeline - Cross account invocation of CodeDeploy for example
* AWS Config - Aggregators
* CloudWatch Events - Event Bus = Multi account events
* Cloudformation - Stack Sets

AWS Control Tower

* Easy way to set up and govern a secure and compliant multi-account AWS environment based on best practices
* Benefits
  + Automate the setup of your environment in a few clicks
  + Automate ongoing policy management using guardrails
  + Detect policy violations and remediate them
  + Monitor compliance through an interactive dashboard
* AWS Control Tower runs on top of AWS Organizations:
  + It automatically sets up AWS Organization to organize accounts and implement SCPs ( Service Control Policies )

AWS Control Tower - Account Factory

* Automates account provisioning and deployments
* Enables you to create pre-approved baselines and configurations options for AWS accounts in your organization ( eg VPC defaul configuration, subnets, regions etc )
* Uses AWS Service Catalog to provision new AWS accounts

AWS Control Tower - Detect and Remediate Policy Violations

* Guardrail
  + Provides ongoing governance for your control tower environment ( AWS accounts )
  + Preventive - Using SCPs
  + Detective - Using AWS Config

Elastic Beanstalk - Saved Configurations

You can make saved configurations based on running envs and it makes it easier for DR

.ebextensioins

EB Extensions files are files that get run by elastic beanstalk to further configure Elastic Beanstalk

Of the configurations which is the order of hierarchy for the application of elastic beanstalk configurations?

* Settings applied directly to the environment
* Saved Configurations
* Configuration Files (ebextentions)
* Default Values

You are additionally able to add any cloudformation to the ebextentions allowing us to add different resources to our beanstalk environment for example a SQS and DynamoDB

AWS EC2 Provisioning - Cloudformation

* KeyName : Name of an existing EC2 KeyPair to enable SSH access to the instance. It must be the name of an existing EC2 KeyPair.
* InstanceType : It is a WebServer EC2 instance type. It must be a valid EC2 instance type.
* SSHLocation : The IP address range that can be used to SSH into the EC2 instances. It must be a valid IP CIDR range of the form x.x.x.x/x.
* HTTPLocation : The IP address range that can be used for HTTP traffic to the EC2 instances. It must be a valid IP CIDR range of the form x.x.x.x/x.
* ICMPLocation : The IP address range that can be used for ICMP traffic to the EC2 instances. It must be a valid IP CIDR range of the form x.x.x.x/x.
* AWSInstanceType2Arch : Provides architecture type of EC2 Instance.
* AWSRegionArch2AMI : Provides the region and AMI details of the EC2 instance.
* EC2Instance : Details of the EC2 instance to be provisioned. It contains InstanceType, SecurityGroups, KeyName, ImageID etc.
* InstanceSecurityGroup : Provides the security group details which will be attached to the EC2 instance.
* Outputs : Once the EC2 is provisioned using this Cloudformation template, parameters found here will be displayed to the user for further use. It includes InstanceId, AZ, PublicDNS, and PublicIP.