API CloudFormation and ECS

CS516 – Cloud Computing

Computer Science Department

Maharishi International University

Maharishi International University - Fairfield, Iowa

All rights reserved. No part of this slide presentation may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying or recording, or by any information storage and retrieval system, without permission in writing from Maharishi International University.

Content

- CloudFormation
- AWS Route 53
- Well Architected

The issue

I make changes to my front-end React application frequently. When I deploy my changes, what I do is:

- Npm install
- Npm run build
- Login to AWS, select region, select service
- Delete existing files in S3 then upload my new build
- Go to cloud front and do invalidation. It takes 20-minute manual work. It is fine, if you do once or twice a month.

CICD - I push my code to GIT and it is deployed. 10 seconds.

- GIT -> Git Actions (pay for minute that it run) -> template

CICD stages

- Install
- Unit test
- Build your app
- Security check on your app build
- Code quality - Sonar (static) or AWS CodeGuru (AI)
- Deployment

Serverless CICD vs Server CICD

- Serverless CICD – Git actions, AWS CodePipeline. It charges how long the build took. It may be too expensive if there are too many builds that run long.
- Server CICD – Jenkins. Server managed by you. Too much work.

Organization has 100 teams. Each team has 10 apps. There are 1000 builds. They all go to this single point (Deployment Servers). Deployment Servers have limit. Maybe there are 4 servers running with 32 GB memory.

We schedule our production deployment at 7 pm. Then start job at 7 pm. IT GOES TO THE QUEUE. You don't know how many minutes you have to wait.

CICD pipeline

After writing your code, there could these actions or stages in the CICD pipeline:

1. Unit tests for your code
2. You can do code quality check (SonarQube - static, CodeGuru - AI)
3. Build (npm run build, docker build t)
4. Do security check for your build (CVE)
5. You can deploy your code to AWS

CloudFormation falls under the last, stage #5.

Tools to implement CICD

Jenkins -> It is nothing but runs the command that you give to it. Here, you need manage a Jenkins **server**.

Downside:

- Jenkins becomes a bottleneck. Because every team and every application is using this one server or cluster. Example, 10:00 am -> start deployment job -> you might need to wait for 10 or 20 minutes. 10:20 am.
- You need an entire team to take care of the Jenkins server.

There are some simple and **serverless** tools:

- Git Actions (free)
- AWS CodePipeline and CodeBuild (pay for the number of build and how long it took)

CloudFormation

AWS CloudFormation is an AWS service that uses template files (JSON, YAML) to automate deployment of AWS resources. Also known as Infrastructure-as-Code (IaC).

- Deployment speed.
- Consistency – you can apply precisely the same configuration repeatedly. In this way, CloudFormation ensures that your applications and services will be consistent and identical, no matter how many instances you create.
- Reduced human errors.
- Easy updates.
- Auditing and change management – Track how resources changed over time.

https://docs.aws.amazon.com/codebuild/latest/userguide/cloudformation-vpc-template.html

https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/CHAP_TemplateQuickRef.html

Template sections

Parameters: Values to pass to your template at runtime (when you create or update a stack). You can refer to parameters from the Resources and Outputs sections of the template.

Mappings: A mapping of keys and associated values. Use the Fn::FindInMap intrinsic function in the Resources and Outputs sections.

Conditions: Conditions that control whether certain resources are created or whether certain resource properties are assigned a value during stack creation or update. For example, you could conditionally create a resource that depends on whether the stack is for a production or test environment.

Resources: Stack resource and their properties.

Outputs: Exporting the resources then other resources can refer it.

Intrinsic functions

Intrinsic functions in CloudFormation:

- Condition – Create resources or set properties based on condition.
- FindInMap – Finds value from 2-level map.
- GetAtt
- ImportValue and Export in the output
- Join
- Split and Select
- Sub
- Ref

You also have intrinsic functions in Step Functions such as generating UUID.

Learn more: Intrinsic function reference

Pseudo parameters reference

You can read these values in your template directly:

- AccountId
- Region
- StackName
- NoValue – Removes the corresponding resource property when specified as a return value in the Fn::If intrinsic function.

You also have similar Pseudo (or Global Context) in Step Functions such as getting current date.

Read more: https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/pseudo-parameter-reference.html

AWS Application Composer

I always wanted a decent tool that helps me to write CloudFormation templates. Finally, that is here. A brand-new service, AWS Application Composer announced in Dec 2022 at AWS re:Invent.

CDK (Cloud Development Kit)

More flexible than CloudFormation templates. You can use programming languages to create resources in the cloud.

Code is synthesized into CloudFormation template. It supports TypeScript.

Learn more: Your first CDK app and Creating your first CDK project

To get started:

- Configure AWS CLI. AWS CDK uses credentials in AWS CLI to create resources.
- Install NodeJS, AWS CDK, TypeScript.
- Bootstrap your account that creates assets that CDK requires when creating resources. cdk bootstrap aws://ACCOUNT-NUMBER/REGION
- cdk init app --language typescript
- npm run build || cdk synth || cdk deploy

Use **CDK**. Automate everything.

Writing CloudFormation template is naïve and doesn’t scale well. Instead, use CDK.

SAM and Amplify are no good.

Route 53

This is a highly available and scalable cloud Domain Name System (DNS) web service.

It is designed to give developers and businesses an extremely reliable and cost-effective way to route end users to applications by translating names, like www.example.com, into the numeric IP addresses, like 98.10.12.31.

An AWS service that allows management of website domains and DNS records.

You can implement canary deployment like Lambda alias at this layer.

Hosted zones

it represents a collection of records that can be managed together, belonging to a single parent domain name. All resource record sets within a hosted zone must have the hosted zone’s domain name as a suffix.

For example, the amazon.com hosted zone may contain records named www.**amazon.com**, and www.aws.**amazon.com**, but not a record named www.amazon.ca.

One hosted zone costs $0.50 per month.

NS records

NS records point to the servers that help to translate domain names into the IP addresses that computers use to communicate with one another.

NS records are automatically created when you create a new hosted zone. Provide that to the domain name provider i.e., GoDaddy. Then you will have full control on your domain name in AWS and create the required records.

Record types

- **Alias** - A type of record that you can create with Amazon Route 53 to route traffic to AWS resources such as ALB, Amazon CloudFront distributions and Amazon S3 buckets.
- **CNAME** - It maps one domain name to another domain name. For example, RDS, ElastiCache.
- **SOA** - The record is created with hosted zone along with NS records. The SOA record stores important information about a domain when the domain was last updated, and how long the server should wait between refreshes.
- **MX** and **TXT** records – Used to setup work email. It is given by the email service provider. You just need to add those records in your hostedzone.

Read more: Supported record types in AWS

Sub domain

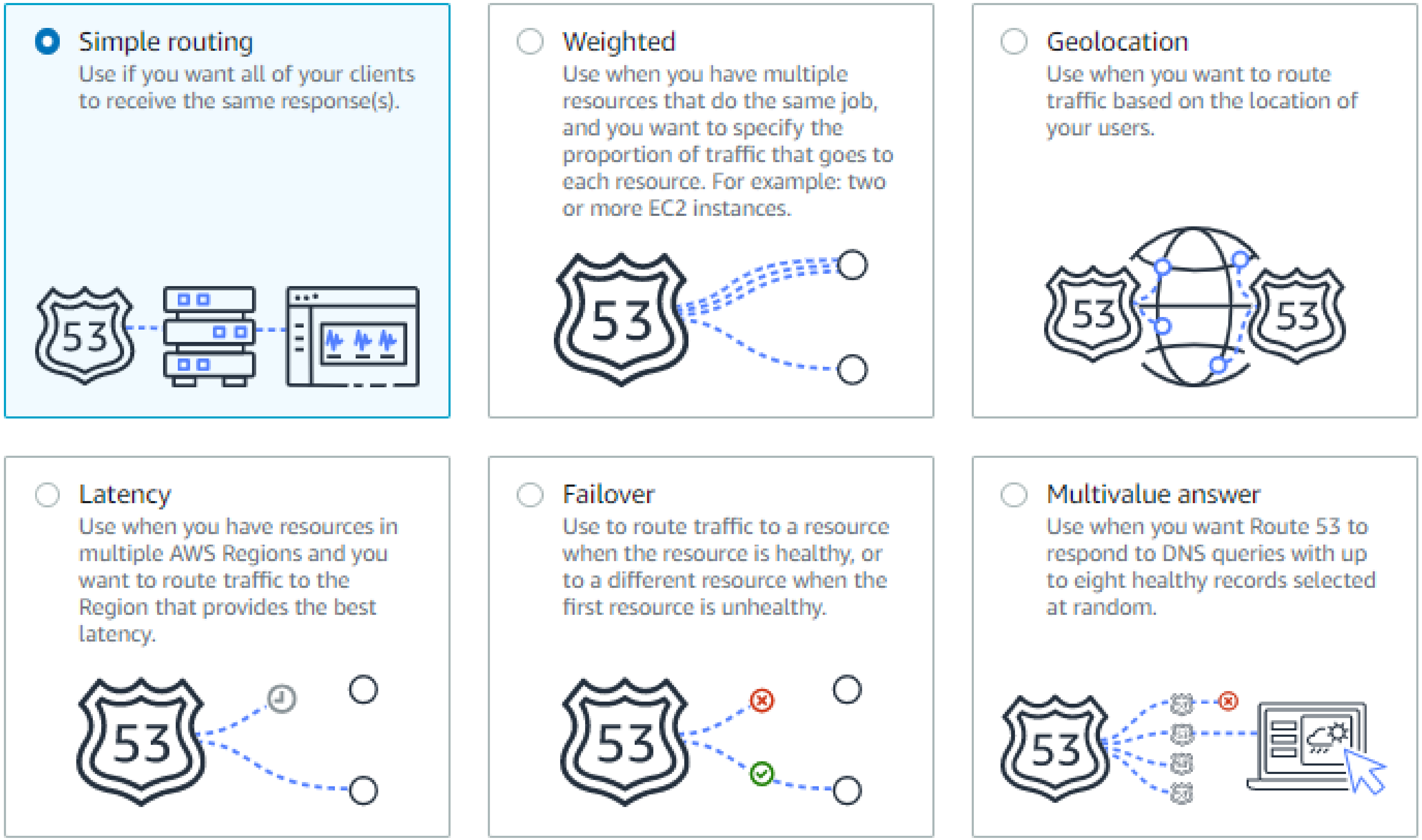
A domain name that has one or more labels prepended to the registered domain name.

For example, The **example.com** domain can have sub domains:

- *accounting*.**example.com**
- *hr*.**example.com**
- *it*.**example.com** so on.

You can create a hosted zone for the sub domain and create, manage its sub domains of the subdomain. For example, **it.example.com**

- *team1.***it**.**example.com**
- *team2.***it**.**example.com**



[✓ Simple routing]
Use if you want all of your clients to receive the same response(s).

Weighted
Use when you have multiple resources that do the same job, and you want to specify the proportion of traffic that goes to each resource. For example: two or more EC2 instances.

Geolocation
Use when you want to route traffic based on the location of your users.

Latency
Use when you have resources in multiple AWS Regions and you want to route traffic to the Region that provides the best latency.

Failover
Use to route traffic to a resource when the resource is healthy, or to a different resource when the first resource is unhealthy.

Multivalue answer
Use when you want Route 53 to respond to DNS queries with up to eight healthy records selected at random.

Routing policies

- **Simple routing policy** – Route internet traffic to a single resource for your domain.
- **Geolocation routing policy** – Use when you want to route internet traffic to your resources based on the location of your users.
- **Latency routing policy** – Use when you have resources in multiple locations and you want to route traffic to the resource that provides the best latency.
- **Failover routing policy** – Failover routing lets you route traffic to a resource when the resource is healthy or to a different resource when the first resource is unhealthy.

Read more: AWS Route 53 routing policies

Time To Live (TTL)

The amount of time, in seconds, that you want a DNS resolver to cache (store) the values for a record before submitting another request to Route 53 to get the current (new) values for that record.

If the DNS resolver receives another request for the same domain before the TTL expires, the resolver **returns the cached value**.

A longer TTL reduces your Route 53 charges, which are based in part on the number of DNS queries that Route 53 responds to.

Amazon Certificate Manager (ACM)

With ACM, you can

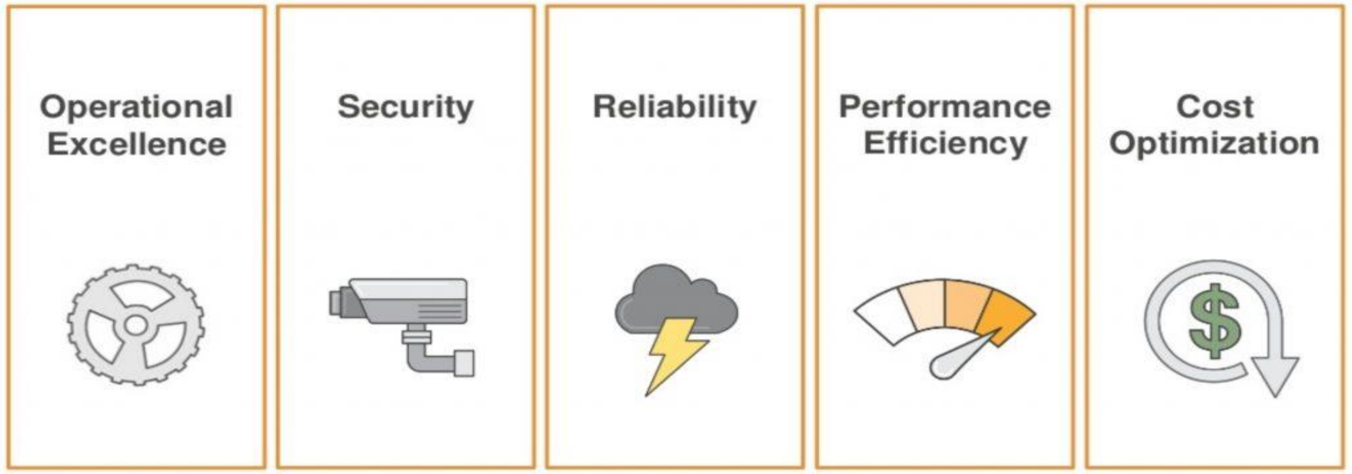
- Create and renew SSL/TLS X.509 certificates for free
- Import third-party certificates into the ACM and use it in your AWS resources by referring its ARN (Amazon Resource Name).

ACM certificates can secure wildcard domains. ACM wildcard certificates can protect an unlimited number of subdomains.

ACM generates a CNAME record that you need to add in the corresponding hosted zone.

AWS Well Architected – 6 pillars

AWS Well-Architected helps cloud architects build secure, high-performing, resilient, and efficient infrastructure for their applications and workloads. The **sustainability** pillar is recently added. That focuses on environmental impacts, especially energy consumption and efficiency.



Read more: AWS academy course and [AWS Well Architected]()

AWS Well Architected Service

- It is a knowledge base that AWS put 20-year knowledge in it.
- When you go to this service on AWS Console, there will be a form that you enter information about your application.
- They evaluate your input or application and then suggest best practices recommendation to you based on the knowledge base.

Operational Excellence pillar - Deliver business value

- Focus - Run and monitor systems to deliver business value, and to continually improve supporting processes and procedures.
- Key topics
  - Automating changes
  - Responding to events
  - Defining standards to manage daily operations
- Principles
  - Perform operations as code
  - Make frequent, small, reversible changes
  - Refine operations procedures frequently
  - Anticipate failure
  - Learn from all operational events and failures

Security pillar - Protect and monitor systems

- Focus - Protect information, systems, and assets while delivering business value through risk assessments and mitigation strategies.
- Key topics
  - Protecting confidentiality and integrity of data (GuardDuty)
  - Identifying and managing who can do what (IAM)
  - Protecting systems
  - Establishing controls to detect security events
- Principles
  - Implement a strong identity foundation
  - Enable traceability
  - Apply security at all layers
  - Automate security best practices
  - Protect data in transit and at rest
  - Keep people away from data
  - Prepare for security events

Reliability pillar - Recover from failure and mitigate disruption.

- Focus - Ensure a workload performs its intended function correctly and consistently when it’s expected to.
- Key topics
  - Designing distributed systems
  - Recovery planning
  - Handling change
- Principles
  - Automatically recover from failure
  - Test recovery procedures
  - Scale horizontally to increase aggregate workload availability
  - Stop guessing capacity
  - Manage change in automation

Performance Efficiency pillar - Use resources sparingly.

- Focus - Use IT and computing resources efficiently to meet system requirements and to maintain that efficiency as demand changes and technologies evolve.
- Key topics
  - Selecting the right resource types and sizes based on workload requirements
  - Monitoring performance
  - Making informed decisions to maintain efficiency as business needs evolve
- Principles
  - Democratize advanced technologies
  - Go global in minutes
  - Use serverless architectures
  - Experiment more often
  - Consider mechanical sympathy

Cost Optimization pillar - Eliminate unneeded expense.

- Focus - Avoid unnecessary costs.
- Key topics
  - Understanding and controlling where money is being spent
  - Selecting the most appropriate and right number of resource types
  - Analyzing spend over time
  - Scaling to meeting business needs without overspending
- Principles
  - Implement Cloud Financial Management
  - Adopt a consumption model
  - Measure overall efficiency
  - Stop spending money on undifferentiated heavy lifting
  - Analyze and attribute expenditure