AWS Certified Solutions Architect– Professional (SAP-C01)

Study Notes

Last updated - Sep'19

AWS Certified Solutions Architect– Professional (SAP-C01) Certification scope

https://d1.awsstatic.com/training-and-certification/docs-sa-pro/AWS%20Certified%20Solutions%20Architect-Professional_Exam%20Guide_2019.pdf

Domain 1: Design for Organizational Complexity

1.1 Determine cross-account authentication and access strategy for complex organizations (for example, an organization with varying compliance requirements, multiple business units, and varying scalability requirements).
1.2 Determine how to design networks for complex organizations
1.3 Determine how to design a multi-account AWS environment for complex organizations

Domain 2: Design for New Solutions

2.1. Determine security requirements and controls when designing and implementing a solution.
2.2. Determine a solution design and implementation strategy to meet reliability requirements.
2.3. Determine a solution design to ensure business continuity.
2.4. Determine a solution design to meet performance objectives.
2.5. Determine a deployment strategy to meet business requirements when designing and implementing a solution.

Domain 3: Migration Planning

3.1. Select existing workloads and processes for potential migration to the cloud.
3.2. Select migration tools and/or services for new and migrated solutions based on detailed AWS knowledge.
3.3. Determine a new cloud architecture for an existing solution.
3.4. Determine a strategy for migrating existing on-premises workloads to the cloud.

Domain 4: Cost Control

4.1. Select a cost-effective pricing model for a solution.
4.2. Determine which controls to design and implement that will ensure cost optimization.
4.3. Identify opportunities to reduce cost in an existing solution.

Domain 5: Continuous Improvement for Existing Solutions

5.1. Troubleshoot solution architectures.
5.2. Determine a strategy to improve an existing solution for operational excellence.
5.3. Determine a strategy to improve the reliability of an existing solution.
5.4. Determine a strategy to improve the performance of an existing solution.
5.5. Determine a strategy to improve the security of an existing solution.
5.6. Determine how to improve the deployment of an existing solution.

Well architected framework

https://d1.awsstatic.com/whitepapers/architecture/AWS_Well-Architected_Framework.pdf

General design principles:

- ❏Stop guessing your capacity needs
- ❏Test systems at production scale:
- ❏Automate to make architectural experimentation easier
- ❏Allow for evolutionary architectures
- ❏Drive architectures using data
- ❏Improve through game days

Oper. Excellence design principles:

- ❏Perform operations as code
- ❏Annotate documentation
- ❏Make frequent, small, reversible changes
- ❏Refine operations procedures frequently
- ❏Anticipate failure
- ❏Learn from all operational failures

Key AWS services:
•CloudFormation, Config, CloudWatch, ElasticSearch (log analysis)

Security design principles:

- ❏Implement a strong identity foundation
- ❏Enable traceability
- ❏Apply security at all layers (sec in depth)
- ❏Automate security best practices
- ❏Protect data in transit and at rest (data classification and add requisite controls)
- ❏Keep people away from data
- ❏Prepare for security events

Key AWS services:
•IAM – AWS IAM
•Detective controls - CloudTrail
•Infra protection – VPC, CF (CDN), Advanced Shield (DDoS), WAF
•Data protection – S3, EBS, EFS, RDS encrypt with KMS
•Incident Response – CWatch Events with lambda

Reliability design principles:

- ❏Test recovery procedures
- ❏Automatically recover from failure
- ❏Scale horizontally to increase aggregate system availability
- ❏Stop guessing capacity
- ❏Manage change in automation

Key AWS services:
IAM , CloudTrail (Change mgmt), VPC,CloudWatch Events, CF, S3, KMS

Perf. Effi design principles:

- ❏Democratize advanced technologies
- ❏Go global in minutes
- ❏Use serverless architectures
- ❏Experiment more often
- ❏Mechanical sympathy

Key AWS services:
•Compute (Auto scaling), Storage – S3, EBS, Db – RDS, DynamoDB, Network – R53, Monitoring – Cwatch, Tradeoffs – Elasticache, CloudFront, Snowball

Cost Optimization design principles:

- ❏Adopt a consumption model
- ❏Measure overall efficiency
- ❏Stop spending money on data center operations
- ❏Analyze and attribute expenditure
- ❏Use managed and application level services to reduce cost of ownership

Key AWS services:
•AWS Cost Explorer, Trusted Advisor, Auto scaling, News blog

Security pillar – consists of following 5 areas

https://d1.awsstatic.com/whitepapers/architecture/AWS-Security-Pillar.pdf

IAM:

Protecting AWS credentials:

- ❏ Protect root users with MFA
- ❏ Use root user for creating additional IAM users which can then assume roles for actions in ur other accounts
- ❏ Use federated identities that is synced with org users, advantage is you dont need to create IAM users and leverage existing credentials and role setup
- ❏ Set strong pwd mgmt policies
- ❏ For CLI and SDK access, create Access key ID and access key
- ❏ For cases where federation or IAM roles is not feasible, use AWS STS to create temp credentials to authenticate to AWS APIs

Fine-grained authorization:

- ❏ Principle of least privilege
- ❏ Fine-grained authorization implemented using IAM roles & policies
- ❏ IAM USER or an AWS SERVICE assumes ROLE which is an IAM PRINCIPLE and is assiged TEMPORARY CREDENTIALS scoped to a SET OF PERMISSIONS. POLICIES are attached to a user, group and roles
- ❏ AWS Organizations lets you centrally manage and enforce policies for multiple AWS accounts.

Detective controls:

Are used to identify a potential security threat or incident

Capture and analyze logs:

- ❏ Use native Apis to collect, filter and analyze logs, no need for a logging infra
- ❏ E.g. use CloudTrail to consolidate all logs from compute, storage and applications to CloudWatch or other log systems
- ❏ For instances based applications, you can use agents for sending logs to Cwatch
- ❏ For Inte Threat detection, Use GuardDuty to to continuously monitor events from CloudTrail, VPC Flow Logs, and DNS logs.
- ❏ Can use Athena for log analytics
- ❏ Use Config for resource inventory, config history and change notifications

Integrate auditing controls with notification and workflow:

- ❏ Integrate the flow of security events and findings into a notification and workflow system such as a ticketing system, a bug/issue system, or SIEM
- ❏ Use CloudWatch events with native or custom event filters to push notifications to lambda, SNS, Slack, OpsGenie ?
- ❏ Use Inspector to do assessment against known sec vulnerabilities & notification

Infrastructure protection:

Protecting network and host-level boundaries

- ❏ Create VPC with private and public subnets
- ❏ Each subnet has attached NACLs (stateless firewall) to restrict traffic e.g. only DB port 3306
- ❏ At host level, configure security groups (stateful firewall)

System security configuration

- ❏ Secure Hosts with threat detection, anti-malware, vulnerability scans
- ❏ Remove need for operator access
- ❏ Use EC2 SSM and CF for automation
- ❏ Use Inspector for scanning

Enforcing service-level protection

- ❏ Protect AWS service endpoints by defining policies using IAM
- ❏ Additionally use resource policies such as with S3 bucket policies and KMS for key mgmt

Data protection:

Data classification

- ❏ Classify based on data location, access levels, data protection (like encryption at rest)
- ❏ Use resource tags, IAM policies, CloudHSM along with d-in-depth

Encryption/tokenization

Protecting data at rest

- ❏ Data in block /object storage, databases, archives, and any other storage medium
- ❏ S3, EBS, RDS allows encryption by choosing your KMS key. S3 also allows storing encrypting objects or upload object with encryption key for on-the-fly encryption

Protecting data in transit

- ❏ Https for Aws service & application invocation. ACM used to manage and deploy public/private certificates

Data backup/replication/recovery

S3 Cross-region replication, EBS snapshots and copy across regions. Glacier for long term backup

Incident response:

Using tags to properly describe your AWS resources, incident responders can quickly determine the potential impact of an incident.

Determine the access your team members need ahead of time, and then regularly verify the access is functional - or easily triggered—when needed.

Use CloudFormation to quickly create a new, trusted environment in which to conduct deeper investigation.

AWS Best Practices for DDoS Resiliency

https://d1.awsstatic.com/whitepapers/Security/DDoS_White_Paper.pdf

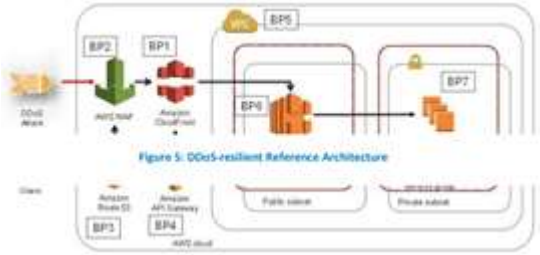


Figure 5: DDoS-resilient Reference Architecture

Infra attacks (Layer 3 and 4 of OSI):

- UDP Reflection attack: aka Src IP Spoofing
  High volume attack with amplification factor
- SYN Flood attack: choke server connections by sending SYN and not sending final Ack
  Low volume attack

Application attacks (Layer 6 and 7 of OSI):

- Http flood attack: S*imulate actual user traffic*
- Cache busting attack: *add variations in query string to disable cache and send traffic to origin server*
- Wordpress XML-RPC flood attack*: link site A to B*
- *Other attacks: DNS atack, TLS attack, scraper bots*

Mitigation against Infra attacks:

★AWS Shield Std – by default enabled for all services and region. Automatically baselines traffic, identifies anomalies and if necessary mitigates against common infra attacks

★Enable CloudFront & R53 – protects with Shield Std, Stateless SYN flood attack, disperse or isolate Attack traffic, Application defense with WAF protection

★Enable Shield Advanced – optional DDoS mitigation service can be applied to CF, R53, CLB, ALB, Elastic Ips has many advanced features , see this link https://aws.amazon.com/shield/

Infra layer defense - BP7 – Resize EC2 compute capacity, choose multiple regions
Infra layer defense - BP6 – Add ALB or NLB load balancers
Infra layer defense - BP1,3 – Use CloudFront and R53, also can protect S3 with R53
App layer defense – BP1,2 – Enable WAF with Web ACLs, Managed rules, rate based rules, enable logging, engage with DRT to enable rules to mitigate attacks, use Firewall manager to manage WAF rules
App layer defense – BP6 – Scale EC2 compute to absorb increased traffic
Obfuscating AWS resources – BP5 - Use EC2 security groups and Network ACLs
Obfuscating AWS resources – BP1,5 – Ensure traffic to origin only comes from CF, use request headers
Obfuscating AWS resources – BP4 – Use Api gateway to protect Apis, recommendation is to regional Api gateway instead of edge optimised as we can use our own CF and enable WAF
Visibility – Use related CloudWatch metrics, Use AWS Shield console, or Api, Use AWS Global Threat Environment dashboard, Use VPC flow logs

Reliability pillar

https://d1.awsstatic.com/whitepapers/architecture/AWS-Reliability-Pillar.pdf

Service Availability - 99,999 %

Availability = Normal Operation Time / Total Time

Availability **decreases** with hard dependencies
invoking system * dependent 1 * dependent 2 = 99.99% * 99.99% * 99.99% = 99.97%

Availability **increases** with redundant components:
maximum availability - ((downtime of dependent 1) * (downtime of dependent 2)) = 100% - (0.1% * 0.1%) = 99.9999%

Application design for High Availability

Fault Isolation zones	Use Availability zones, Use AWS regions
Redundant components	AWS internally uses redundant components to be resilient to failure of a single compute node, single storage volume, or single instance of a database
Micro-service architecture	You know
Recovery Oriented Computing	Focus on right mechanisms to detect failures (such as ELB or Route53 health checks). After a failure occurs ROC would apply one of a small number of well-tested recovery paths. Use common recovery path. Eg. auto scale for any instance failure Testing recovery paths is imp.
Distributed systems best practices	Throttling,Retry with exponential backoff, Fail fast, Idempotency tokens, Constant work, Circuit breaker, Bi-modal behaviour

Operational considerations for Availability:

- ❏Automate Deployments to Eliminate Impact - Canary, BG, Feature toggles, Failure isolation zone deployments
- ❏Testing - perf, simian army
- ❏Monitoring & Alarming - measure latency, errors & availability, Measure Percentile, user tasks testing

Generation
- ❏Identify which services to monitor and which metrics

Aggregation
- ❏Event Logs are aggregated to CloudWatch then to 3rd party systems

Real-time processing & Alarming
- ❏Use SQS, SNS, lambda, Opsgenie, Slack

Storage & Analytics
- ❏CloudWatch to S3, or 3rd party Splunk, NewRelic, loggly etc.

Operational Readiness Review (ORR) - use ORR checklist
Audit - CloudWatch logs, Config, CloudTrail

Reliability pillar - Example implementations for diff Availability goals

https://d1.awsstatic.com/whitepapers/architecture/AWS-Reliability-Pillar.pdf

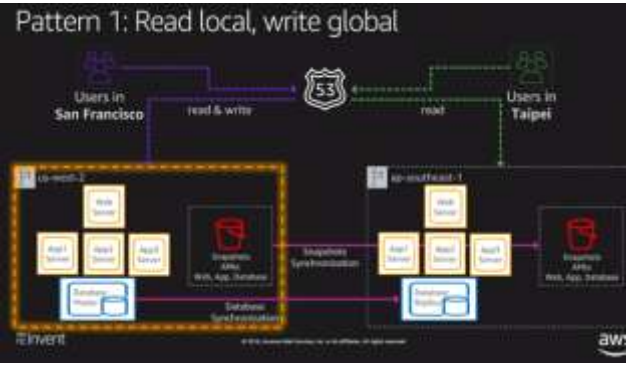
Availability ->	99.9% (8 hrs 45 m) Single region	99.99% (52 mins) Single region	99.95% (4 hrs 22 m) RTO < 30m Multi region	99.999% or higher (5 m) Multi region
Adapting to changes in demand	ELB for web and auto scaling application tier; resizing Multi-AZ RDS		*In addtion>* this is synchronized between AWS Regions for static stability	
Monitoring	Site health check only; alerts sent when down	Health checks at all layers and on KPIs; alerts sent when configured alarms are tripped; alerting on all failures. Operational meetings are rigorous to detect trends and manage to design goals.	*In addtion>* Health checks at all layers, including DNS health at AWS Region level, and on KPIs;	
Deploying changes	Automated deploy in place and runbook for rollback.	Automated deploy via canary or blue/green and automated rollback when KPIs or alerts indicate undetected problems in application. Deployments are made by isolation zone.	*In addtion>* deployments are made to one isolation zone in one AWS Region at a time.	
Backups	Automated backups via RDS to meet RPO and runbook for restoring.	Automated backups via RDS to meet RPO and automated restoration that is practiced regularly in a game day.	*In addtion>* Automated backups in each AWS Region	
Implementing resiliency	Auto scaling to provide self-healing web and application tier; RDS is Multi-AZ.	Implemented fault isolation zones for the application; auto scaling to provide self-healing web and application tier; RDS is Multi-AZ.	*In addtion>* Regional failover is managed manually with static site presented while failing over.	*In addtion>* *Regional failover automated*
Testing resiliency	ELB and application are self-healing; RDS is Multi-AZ; no explicit testing.	Component and isolation zone fault testing is in pipeline and practiced with operational staff regularly in a game day; playbooks exist for diagnosing unknown problems; and a Root Cause Analysis process exists	*In addtion>* with communication paths for what the problem was, and how it was corrected or prevented.	*In addtion>* RCA correction is prioritized above feature releases for immediate implementation and deployment.
Disaster recovery	Encrypted backups via RDS to same AWS Region.	Encrypted backups via RDS to same AWS Region that is practiced in a game day.	Encrypted backups via RDS, with replication between two AWS Regions. Restoration is to the current active AWS Region, is practiced in a game day, and is coordinated with AWS.	

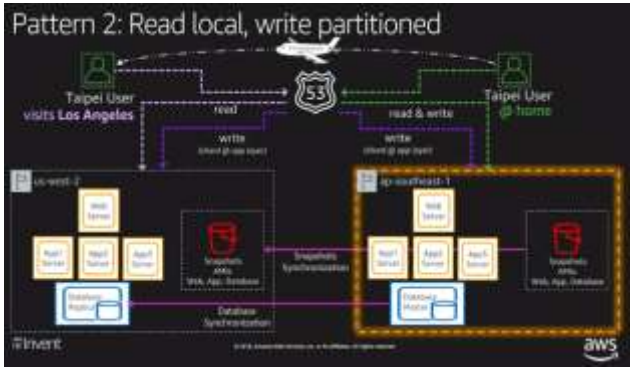
Disaster Recovery

Configuration	Data loss (RPO)	Recovery time (RTO)
Backup & Restore	Few Hours	< 24 hrs
Pilot Light	Few Minutes	< 1 hour
Warm standby	< 1 second	Few minutes
Hot standby (Active-Active)	Zero data loss	Instantaneous

Depending on RTO and RPO, AWS offers following 4 basic techniques for back-up and disaster recovery

- ***Backup and Restore*** - ideal for non-critical and applications and can endure higher degree of data loss and time for recovery. Back up application state into S3 or glacier, and then use the region replication feature of S3 to replicate to another region. In this scenario, data loss and recovery time is highest.
- ***Pilot Light*** - Recovery time is less than Backup & Restore method. Data loss is in range of minutes and recovery time is in range of hours. Here the core piece of the system such as database is kept running in the secondary region and data replication is enabled. Server images are created and updated periodically and instances created by CF templates
- ***Warm standby*** - For applications that are critical and can endure medium degree of data loss and time for recovery. Describes a DR scenario in which a scaled-down version of a fully functional environment is always running in a different AWS region
- ***Active-Active across regions*** - Applicable for applications that are highly critical and can endure only least degree of data loss (RPO < 5 min) and minimal time for recovery (RTO < 5 min).





Web Application Hosting in AWS

https://d0.awsstatic.com/whitepapers/aws-web-hosting-best-practices.pdf

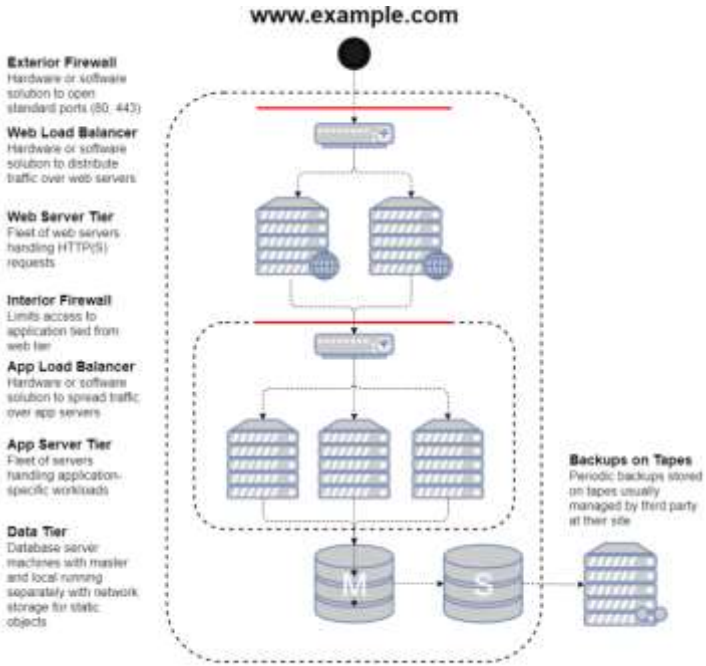


Figure 1. A traditional web hosting architecture

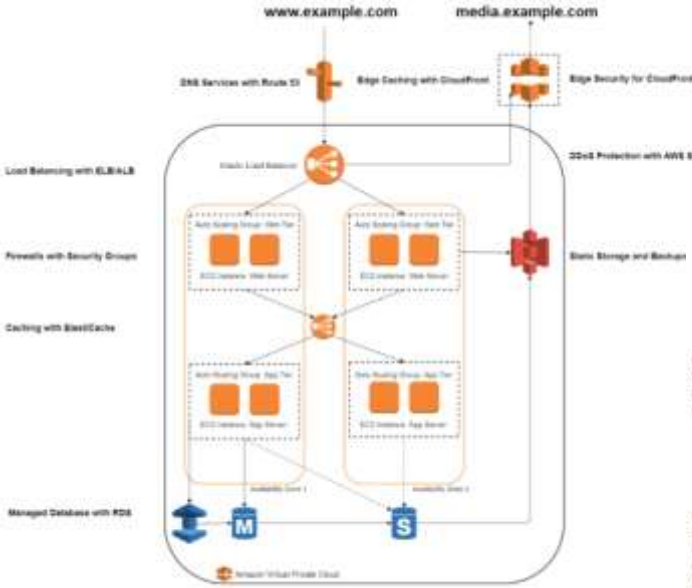


Figure 3. An example of a web hosting architecture on AW

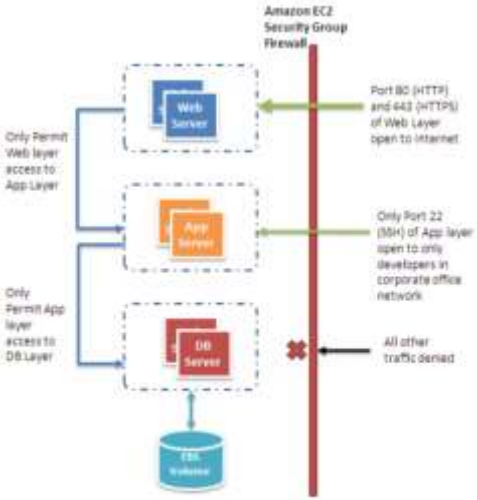
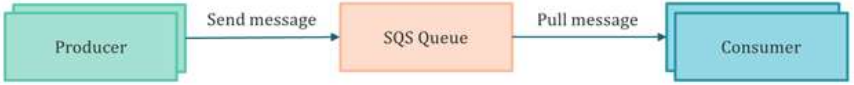


Figure 4. Security groups in a web application

Application Integration

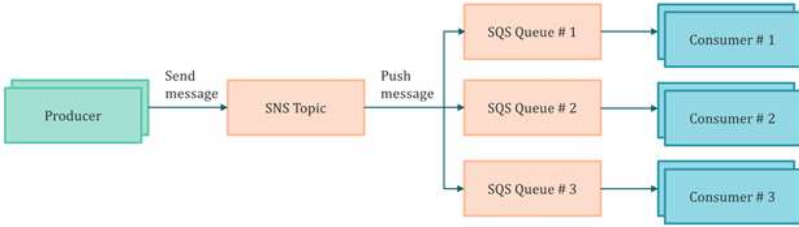
SQS, SNS

SQS/SNS



SQS offers two types of message queues:

- Standard queues offer maximum throughput, best-effort ordering, and at-least-once deliver
- FIFO queues are designed to guarantee that messages are processed exactly once, in the exact order that they are sent, with limited throughput.



Key Design decisions & best practices

- Use Standard queue when high throughput is more important than message ordering
- Use FIFO queue when ordering is more important than throughput. It has throughput limit of 300 / 3000 TPS without and with batching respectively. To enable exactly-once processing, use content based or message deduplication Id
- If using standard queues, delete each message from queue after consuming the same. If this is not done, message will again be eligible for consumption after the visibility timeout period.
- SQS does not support multiple consumer applications reading the same set of messages from the same queue. Create a SNS topic and add appropriate subscribers for fanout scenario where messages need to be pushed to multiple subscribers.
- Prefer long polling to short polling as it results in higher performance at reduced cost in most cases.
- Message size is limited to 256KB (body & attributes). To process larger messages upto 2GB, use SQS along with Amazon S3.
- Use Dead letter queues to isolate and diagnose messages that can't be processed correctly
- Use Delay queues when you want to postpone delivery of new messages in queue. Currently delay can be set between 0 and 15 minutes
- SQS uses HTTP REST protocol and a proprietary SDK. However, Amazon does offer a JMS implementation of the SQS SDK.

Service High Availability

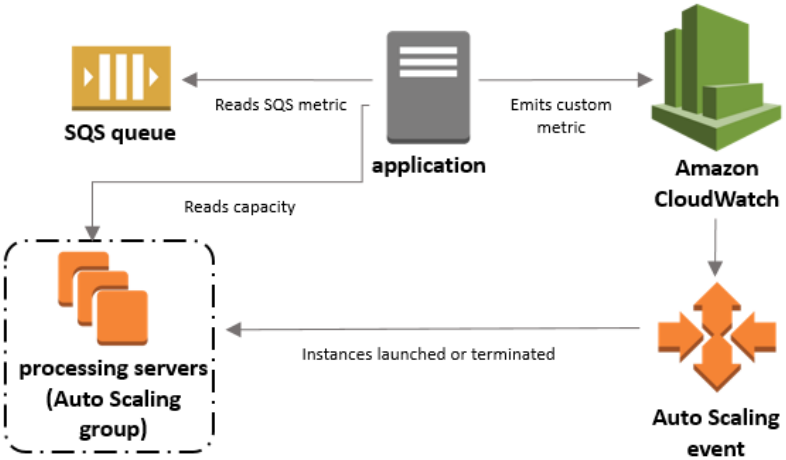
SQS is a highly available and distributed service which makes it reliable. Messages are replicated inside AWS, making message loss due to node failure virtually non-existent. But SQS is a regional service. There is no built-in multi-region availability method for guarding against region failure. If a region fails, your topic or queues will be inaccessible till the service is up. There isn't any standard way to handle them in a multi-region fashion.

Scalability

Scaling is achieved by increasing the number of message producers (making SendMessage requests) and consumers (making ReceiveMessage and DeleteMessage requests) in order to increase the overall queue throughput. You can scale horizontally by increasing the number of threads on a client, adding clients, or both.

Scaling EC2 based on SQS

https://docs.aws.amazon.com/autoscaling/ec2/userguide/as-using-sqs-queue.html



Define parameters that control the scaling process. For example, you can create a policy that calls for enlarging your fleet of EC2 instances whenever the average number of messages reaches a certain level. This is useful for scaling in response to changing conditions, when you don't know when those conditions will change.There are 3 main parts to this confign:

- An Auto Scaling group to manage EC2 instances for the purposes of processing messages from an SQS queue.
- A custom metric to send to Amazon CloudWatch that measures the number of messages in the queue per EC2 instance in the Auto Scaling group.
- A target tracking policy that configures your Auto Scaling group to scale based on the custom metric and a set target value. CloudWatch alarms invoke the scaling policy.

Choosing an Effective Metric and Target Value

The number of messages in your SQS queue does not solely define the number of instances needed. In fact, the number of instances in the fleet can be driven by multiple factors, including how long it takes to process a message and the acceptable amount of latency (queue delay).The solution is to use a ***backlog per instance* metric** with the target value being the *acceptable backlog per instance* to maintain.

To illustrate, the current ApproximateNumberOfMessages is 1500 and the fleet's running capacity is 10. If the average processing time is 0.1 seconds for each message and the longest acceptable latency is 10 seconds, then the acceptable backlog per instance is 10 / 0.1, which equals 100. This means that 100 is the target value for your target tracking policy. Because the backlog per instance is currently at 150 (1500 / 10), your fleet scales out by five instances to maintain proportion to the target value.