AWS SERVERLESS SERIVCES

14 March 2024 14:27

1. Introduction to Serverless Computing:
   - Serverless computing is an extension of cloud computing where developers write code without worrying about managing underlying servers.
   - It eliminates the need for managing infrastructure, allowing developers to focus solely on writing code.
2. Evolution of Computing:
   - Started with physical data centers, then moved to virtualization, and eventually to the cloud.
   - Serverless computing is the next step, where even the management of cloud infrastructure is abstracted away.
3. Why Go Serverless?:
   - Simplifies development by eliminating infrastructure management.
   - Event-driven architecture ensures that compute resources are only used when needed, reducing costs.
   - Pay-as-you-go billing model charges only for the time the code is running, providing cost efficiency.
4. Embracing Serverless Computing:
   - AWS provides two major serverless compute options: Lambda and Fargate.
   - Lambda allows developers to write code and define triggers without managing servers.
   - Fargate enables running containers without managing underlying infrastructure.
5. Exam Tip:
   - Embrace serverless architecture and favor managed services like Lambda and Fargate over traditional EC2 instances whenever possible.
   - Focus on using more managed tools to offload infrastructure management to AWS.

   Understanding serverless computing and leveraging managed services like Lambda and Fargate can lead to more efficient and cost-effective application development.

From <https://chat.openai.com/c/d6fe90a1-a00b-4374-9bfc-54c5891880f3>

1. What is AWS Lambda?:
   - AWS Lambda is a serverless compute service that allows you to run code without provisioning or managing servers.
   - It enables running code on demand with automated scaling, freeing developers from server-related overhead.
2. Lambda Features:
   - Lambda offers a free tier including a certain number of invocation requests and compute memory per month.
   - Seamlessly integrates with various AWS services like S3, DynamoDB, EventBridge, SQS, SNS, and Kinesis.
   - Provides logging and monitoring through Amazon CloudWatch.
   - Allows configuring memory requirements, CPU scales proportionally with memory, and has a maximum timeout of 15 minutes.
3. Building a Lambda Function:
   - Key configurations include selecting the runtime environment, defining permissions using IAM roles, configuring networking, setting resource allocations (memory and timeout), and specifying triggers for function invocation.
4. Quotas and Limitations:
   - Lambda has limitations on concurrent executions, disc storage, environment variables size, memory allocation, and function duration.

- Deployment package size, request/response payload size, and streaming response size also have restrictions.

5. Popular Architecture Examples:
   - Example architectures include processing data files uploaded to S3 using Lambda triggers and performing scheduled tasks using Amazon EventBridge to trigger Lambda functions.
6. Takeaways:
   - Lambda is useful for automating operational tasks and responding to events or conditions.
   - Understanding Lambda's limitations, triggers, and integration with other services is crucial.
   - Lambda excels in running small, short, and lightweight functions, making it ideal for various tasks.
   - It can run both inside and outside a custom VPC, with VPC integration needed for accessing private resources like RDS instances.

   Overall, Lambda is a powerful tool for building serverless applications and automating tasks within the AWS ecosystem. Understanding its features, configurations, limitations, and integration options is essential for leveraging its full potential.

From <https://chat.openai.com/c/d6fe90a1-a00b-4374-9bfc-54c5891880f3>

1. What is the AWS Serverless Application Repository?:
   - SAR is a service that allows developers and organizations to find, deploy, and publish serverless applications within their AWS accounts.
   - Applications are defined using AWS SAM templates (Serverless Application Model templates) along with the application code.
   - SAR is deeply integrated within the AWS Lambda service and appears within the Lambda console.
2. Publishing and Deploying Applications:
   - Publishing: When you publish an application, you make it available for others to find and deploy. Published apps are initially set to private and can be shared with specific accounts or publicly.
   - Deploying: With deployment, you can find and deploy published applications from the console or the public catalog. Deployments can be done with a few clicks, making it easy to deploy entire serverless applications.
3. Exam Tips:
   - Understand SAM templates: SAR applications are defined using SAM templates, which are essentially CloudFormation templates.
   - Know the difference between publishing and deploying: Publishing makes your app available for others, while deploying allows you to deploy published apps.
   - Recognize the integration with AWS Lambda: SAR is integrated within the Lambda console, allowing easy deployment of serverless applications.

From <https://chat.openai.com/c/d6fe90a1-a00b-4374-9bfc-54c5891880f3>

1. Container Definition:
   - A container is like a normal container in daily life; it holds things. Similarly, in IT, a container holds everything needed to run an application, including code, dependencies, and configuration files.
   - Containers offer portability, allowing the same environment to be moved anywhere: development, production, or staging.
2. Advantages of Containers:
   - Unlike virtual machines, containers don't duplicate the entire guest operating

system, leading to efficient resource utilization.
    - Containers create a microenvironment tailored to the application's needs, eliminating unnecessary components.
    - They provide a better "bang for the buck" by maximizing resource usage.
3. Terminology:
    - Dockerfile: Instructions file to build a container image.
    - Image: Immutable template containing everything needed to run an application.
    - Registry: Repository for storing and sharing container images.
    - Container: Running instance of an image.
4. Building and Running a Container:
    - Demonstrated creating a Dockerfile with instructions.
    - Built an image using the Dockerfile.
    - Ran the image as a container, resulting in a running website accessible via port 80.
5. Exam Tips:
    - Containers are seen as flexible and portable solutions, providing control over the application environment.
    - Understand the use cases for containerization: portability, ease of use, and flexibility.
    - Focus on high-level concepts rather than specific details like individual lines in a Dockerfile.
    - Recognize the importance of consistency between development and production environments facilitated by containers.
6. Closing Remark:
    - Remember to take a picture inside your refrigerator before shopping to avoid forgetting what you already have—a metaphor for maintaining consistency between environments using containers.

From <https://chat.openai.com/c/d6fe90a1-a00b-4374-9bfc-54c5891880f3>

1. Problems with Containers:
    - At scale, managing containers can become complex and inefficient, leading to administrative overhead and potential cost issues.
2. Amazon ECS:
    - ECS is a managed service for deploying and managing Docker containers on AWS.
    - It can handle anywhere from one to thousands of containers, automatically placing and managing them on appropriate hosts.
    - ECS integrates with load balancers for efficient traffic distribution, and it supports IAM roles for enhanced security.
    - It's easy to set up and scale to meet workload demands.
3. Other Container Orchestration Methods:
    - Kubernetes: An open-source container orchestration platform suitable for large-scale, complex operations. AWS offers a managed version called Amazon EKS.
    - Kubernetes can be used on-premises or in the cloud, providing flexibility but requiring more configuration and integration effort compared to ECS.
4. Choosing ECS or EKS:
    - ECS is best suited for AWS-specific environments, offering simplicity and ease of use.
    - EKS is suitable for scenarios where you're not fully committed to AWS and for larger workloads. It's more complex to configure but provides greater flexibility.
5. Exam Tips:
    - Consider using ECS for simple container orchestration within AWS unless Kubernetes is explicitly mentioned.
    - Favor AWS services over third-party solutions, and consider Amazon EKS for

Kubernetes-related scenarios.
- Understand that ECS and EKS are suitable for long-running applications, such as web applications that require continuous availability.

With these concepts in mind, you're better prepared to choose between Amazon ECS and Amazon EKS for your container orchestration needs.

From <https://chat.openai.com/c/d6fe90a1-a00b-4374-9bfc-54c5891880f3>

1. AWS Fargate:
   - Fargate is a serverless compute engine for running Docker containers, fully managed by AWS.
   - It allows developers to focus on application development without worrying about underlying infrastructure management.
   - Fargate can run both Linux and Windows containers, providing flexibility for various application needs.
   - It requires the use of either Amazon ECS or Amazon EKS to orchestrate and deploy containers.
2. Comparing ECS Launch Types:
   - ECS offers two launch types: EC2 and Fargate.
   - EC2 launch type requires managing underlying instances and follows the EC2 pricing model, suitable for long-running containers.
   - Fargate launch type abstracts underlying infrastructure, follows a serverless pricing model, and is ideal for shorter running tasks with isolated environments per container.
3. Choosing Fargate over Lambda:
   - Fargate is suitable for consistent and predictable workloads, providing greater control over Docker containers.
   - Lambda is ideal for event-driven, serverless computing, scaling automatically to handle changes in demand, and is designed for single-stateless functions with limited execution time.
4. In-Console Demonstration:
   - Demonstrated creating a task definition and a Fargate cluster in ECS.
   - Created a service using Fargate launch type to deploy a web server container.
   - Showed how Fargate abstracts underlying infrastructure management, making it easier to deploy and run containers.
5. Exam Tips:
   - Understand factors influencing the choice between Fargate, Lambda, and EC2 for containerized workloads.
   - Recognize Fargate as a serverless compute option, available through ECS or EKS, and its benefits of abstracting infrastructure management.
   - Be aware of the trade-offs between Fargate and other launch types, considering factors like user-friendliness and cost.
   - Know the use cases for Fargate versus Lambda and understand IAM permissions using Task Roles for containers.

With these concepts and exam tips, you're better prepared to choose and utilize Fargate for containerized applications in AWS.

1. Overview of Amazon EventBridge:
   - EventBridge is a fully managed serverless event bus service provided by AWS.
   - It acts as a central nervous system for serverless applications, allowing you to build event-driven architectures.
   - EventBridge enables you to pass events from various sources to different endpoints and connect different parts of your serverless apps together.

2. Important Concepts:
    - Events: Recorded changes or occurrences within an AWS environment, a SaaS partner, or your own applications and services.
    - Rules: Criteria that define how incoming events are processed, matching events based on event patterns or schedules.
    - Event Bus: Receives events from various sources and delivers them to intended targets or destinations, such as Lambda functions or SNS topics.
3. Rule Triggers:
    - Event Patterns: Define specific conditions for triggering rules based on real-time events, specifying event sources and patterns.
    - Scheduled Rules: Set up recurring schedules for triggering rules at specified intervals using rate-based or cron expressions.
4. Architectures and Use Cases:
    - Example architecture leveraging EventBridge: Detecting instance state changes in EC2 instances and triggering actions such as sending SNS notifications and invoking Lambda functions for automated remediation.
5. In-Console Demo:
    - Demonstrated creating an EventBridge rule to detect EC2 instance shutdown events and trigger SNS notifications and Lambda function invocations.
    - Showed how to configure event patterns, targets, and additional settings for rules.
6. Exam Tips:
    - Understand when to use EventBridge for triggering actions or workflows in response to events in AWS environments.
    - Know the different rule configurations based on event patterns or schedules and how to customize them.
    - Recognize EventBridge as the successor to CloudWatch Events and understand its role in building event-driven architectures.
    - Be aware of the default event bus provided by AWS and the option to create custom event buses for organizing and managing events.

    By understanding these concepts and tips, you'll be better prepared to utilize EventBridge for event-driven architectures and workflows in AWS

1. Service Overview:
    - Amazon ECR is a managed container image registry service provided by AWS.
    - It supports private repositories with resource-based permissions using AWS IAM.
    - Users can push and pull images in Open Container Initiative (OCI) or Docker format to and from ECR.
2. Components:
    - Registry: A regional private registry provided to each AWS account for storing container images.
    - Authentication Token: Used to authenticate users for pushing and pulling images.
    - Repositories: Containers for storing Docker images, OCI images, or OCI artifacts, each with its own access policy.
    - Image: The container image stored in a repository.
3. Features:
    - Lifecycle Policies: Define rules for cleaning up unused images in repositories.
    - Image Scanning: Identifies software vulnerabilities within container images and generates reports.
    - Sharing: Supports cross-region and cross-account sharing configurations.
    - Cache Rules: Enables pull-through caching for public repositories.

- Tag Mutability: Prevents image tags from being overwritten, configured on a per-repository level.

4. Integrations:
   - Integrates with Amazon ECS for using images in container definitions.
   - Integrates with Amazon EKS for pulling images for Kubernetes clusters.
   - Also compatible with Amazon Linux containers for local software development.
5. Exam Tips:
   - Understand Amazon ECR as a managed container image registry supporting various image formats.
   - Know how to implement lifecycle policies and image scanning for security.
   - Familiarize yourself with tag mutability and its role in preventing tag overwriting.
   - Look out for keywords related to managed container image registry, OCI compatibility, and integrations with ECS and EKS.

   By mastering these concepts, you'll be well-prepared to leverage Amazon ECR for storing custom Docker images securely and efficiently in your AWS environment.

From <https://chat.openai.com/c/d6fe90a1-a00b-4374-9bfc-54c5891880f3>

1. Overview:
   - EKS-D is a Kubernetes distribution based on and used by Amazon EKS.
   - It has the same versions and dependencies deployed by Amazon EKS.
   - Unlike Amazon EKS, EKS-D is fully managed by users, allowing deployment anywhere - on-premises, in the cloud, or elsewhere.
2. Exam Tips:
   - Look out for questions related to self-managed Kubernetes deployments similar to Amazon EKS.
   - EKS-D is used when you need to run versioned deployments of clusters outside of AWS-managed services while maintaining the same dependencies and requirements.
   - While not commonly appearing on exams, understanding EKS-D is beneficial, especially for scenarios requiring self-managed Kubernetes deployments.

   By understanding Amazon EKS Distro, you'll be prepared to deploy and manage Kubernetes clusters independently, ensuring compatibility with Amazon EKS while having full control over the deployment environment.

1. Amazon EKS Anywhere:
   - It enables running and managing on-premises EKS Kubernetes clusters based on the EKS Distro.
   - EKS Anywhere provides full lifecycle management for on-premises Kubernetes clusters independently of AWS services.
   - Control plane management is entirely done by customers, and updates are performed manually via CLI or Flux.
   - It offers curated packages for extending core functionalities of on-premises Kubernetes clusters, available via enterprise subscription.
2. Amazon ECS Anywhere:
   - ECS Anywhere allows running and managing container applications on-premises, including on VMs and bare-metal servers.
   - It offers a fully managed solution for standardizing container management across environments based on ECS standards and practices.
   - Requirements include installing and configuring SSM Agent, ECS Agent, and Docker on local servers, and registering external instances as Systems Manager-

managed instances.
    - Usage involves executing installation scripts on external instances and leveraging the EXTERNAL launch type within ECS to manage clusters from the AWS console.
3. Exam Tips:
    - Understand the concepts and differences between EKS Anywhere and ECS Anywhere.
    - Remember that EKS Anywhere is based on the EKS Distro and requires full lifecycle management by customers.
    - Know that ECS Anywhere is a feature within Amazon ECS itself, offering fully managed container orchestration anywhere, with specific requirements for installation and usage.

By understanding EKS Anywhere and ECS Anywhere, you'll be prepared to deploy and manage containers outside AWS environments while leveraging AWS-managed services and standards.

From <https://chat.openai.com/c/d6fe90a1-a00b-4374-9bfc-54c5891880f3>

1. Overview:
    - Amazon Aurora Serverless is an auto-scaling configuration for Aurora databases, adjusting capacity based on application demand.
    - It offers per-second billing, charging only for resources consumed, making it cost-effective.
    - The service is ideal for workloads with variable activity, multi-tenant apps, new applications, development and testing, mixed-use applications, and capacity planning.
2. Concepts:
    - Aurora Capacity Units (ACUs) measure cluster scaling, with a minimum and maximum capacity set by users.
    - ACUs are quickly allocated from AWS-managed warm pools, ensuring rapid scaling based on demand.
    - Each ACU combines memory, CPU, and networking capabilities, with data resiliency similar to Aurora Provisioned.
    - High availability is achieved through Multi-AZ deployments.
3. Use Cases:
    - Variable workloads with unpredictable activity.
    - Multi-tenant applications where individual database capacity management is impractical.
    - New applications where database sizing is uncertain.
    - Development and testing environments requiring flexible scaling.
    - Mixed-use applications with varying traffic patterns.
    - Capacity planning to optimize database capacity for workloads.
4. Exam Tips:
    - Understand that Amazon Aurora Serverless is an on-demand, auto-scaling, serverless version of Aurora.
    - Remember the concept of ACUs and how they determine cluster scaling.
    - Recall that billing is based on per-second usage of resources.
    - Be familiar with common use cases such as variable workloads, new applications, development and testing, and capacity planning.
    - Know that Aurora Serverless offers the same data resiliency and high availability options as Aurora Provisioned.

By mastering these concepts and tips, you'll be well-prepared to leverage Amazon Aurora Serverless for scalable, cost-effective database solutions

1. Overview:
    - AWS X-Ray collects data about requests served by your application, including calls

to downstream resources like microservices, APIs, or databases.
   - It helps you visualize, filter, and gain insights into your application's performance and behavior.
   - Traces are collected from your application and used to provide insights, with many AWS services able to automatically provide traces to X-Ray.
2. Concepts:
   - Segments contain data about requests, including resource names and request details.
   - Subsegments provide granular timing information about application calls.
   - Service graph visually represents interactions between services within requests.
   - Traces provide a trace ID to track request paths and collect all segments within a request.
   - Tracing header (X-Amzn-Trace-Id) contains sampling decisions and the original trace ID for requests, helping to manage tracing costs.
3. X-Ray Daemon:
   - X-Ray Daemon collects raw segment data from applications and sends it to the X-Ray API.
   - It listens on UDP port 2000 and works alongside X-Ray SDKs to make collecting trace data easier.
4. Integrations:
   - Integrates with EC2 instances, ECS tasks, Lambda functions, Elastic Beanstalk environments, API Gateway stages, SNS, and SQS.
   - Can be installed as a Daemon on EC2 instances or added as a configuration option for various AWS services.
5. Exam Tips:
   - Understand that AWS X-Ray provides insights into application requests and responses.
   - Be familiar with terms like traces, tracing headers, and segments.
   - Know how to integrate X-Ray with various AWS services and how to toggle it on or off.
   - Look out for exam scenarios involving application request insights, downstream resource response times, and HTTP response analysis.

   By mastering these concepts and tips, you'll be well-prepared to use AWS X-Ray for gaining insights into your application's performance and behavior

From <https://chat.openai.com/c/d6fe90a1-a00b-4374-9bfc-54c5891880f3>

1. Overview:
   - AWS AppSync is a robust and scalable GraphQL interface used by application developers.
   - It enables developers to combine data from multiple sources like Amazon DynamoDB and AWS Lambda.
   - GraphQL is a data language that allows applications to fetch data from servers, encouraging declarative coding and working seamlessly with modern tools and frameworks.
2. Exam Tips:
   - Understand that AWS AppSync is a scalable GraphQL interface primarily targeted towards application developers, especially frontend development.
   - Look out for keywords like GraphQL, data fetching, declarative coding, and frontend application development, as these are scenarios where AWS AppSync might be a suitable solution.

   By being familiar with AWS AppSync and its use cases, you'll be prepared to recognize when it's appropriate to use this service in application development scenarios.

From <https://chat.openai.com/c/d6fe90a1-a00b-4374-9bfc-54c5891880f3>

1. Questions to Ask Yourself:

- Consider if the application is suitable for containers or if it should be hosted on EC2.
- Determine if the application is AWS-specific and if services like ECS or Lambda can be leveraged.
- Assess the runtime requirements of the code to choose the appropriate AWS service.

2. AWS Lambda:
   - Utilize IAM roles for Lambda functions to access AWS resources securely.
   - Understand Lambda triggers, limitations, and the ability to make AWS API calls.
3. Containers and Images:
   - Consider Kubernetes for open-source solutions and Amazon EKS or EKS Anywhere for AWS-managed container orchestration.
   - Remember that Fargate requires ECS or EKS to work and offers flexibility for various workloads.
   - Know the basics of container workflow and consider Amazon ECR for AWS-managed image registry.
4. Amazon Aurora Serverless:
   - Recognize it as an on-demand and auto-scaling database suitable for variable workloads and capacity planning.
5. AWS X-Ray:
   - Understand its use for gaining application insights, tracing requests and responses, and its tight integration with Lambda and API Gateway.
6. AWS AppSync:
   - Recognize it as a managed GraphQL interface suitable for front-end application development.

   By understanding these key points and considering them in different scenarios, you'll be well-prepared to tackle questions related to serverless architectures in the exam

From <https://chat.openai.com/c/d6fe90a1-a00b-4374-9bfc-54c5891880f3>