**Designing Cost-Optimized Compute**

**Application and deployment services**

* **Lambda**: AWS Lambda lets you run code without provisioning or managing servers. You only pay for the compute time you consume.
* **API Gateway**: This service allows you to create, publish, maintain, monitor, and secure APIs at any scale.
* **Kinesis**: AWS Kinesis makes it easy to collect, process, and analyze real-time, streaming data.
* **CloudFront**: A content delivery network (CDN) that securely delivers data, videos, applications, and APIs to customers globally with low latency.
* **OpsWorks**: A configuration management service that provides managed instances of Chef and Puppet.

**Additional application services**

* **AWS Amplify**: This is a full-stack deployment tool for both front-end and back-end. It supports web and mobile applications, including features like authentication, storage, and APIs. It integrates with frameworks like Angular, Vue, React, and Next.js. Amplify also supports visual UI design with Figma, converting designs into code, and can connect to external repositories like Git.
* **AppSync**: This service provides a GraphQL-based solution for data integration. It allows you to access multiple databases through a single interface. AppSync acts as a proxy, enabling access to data sources like DynamoDB, Lambda, and HTTP APIs, even if they don't natively support GraphQL.
* **Device Farm**: This service offers a collection of real iOS and Android devices for testing purposes. You can perform automated testing using frameworks like Appium, Calabash, and UI Automator, or manual testing with remote access and control. This is useful for testing mobile applications without needing to maintain your own pool of devices.

These services enhance the AWS ecosystem by providing powerful tools for deploying and managing applications.

**Lambda**

* **AWS Lambda**: This is a compute service that allows you to run code without provisioning or managing servers. You simply upload your code, and Lambda takes care of everything required to run and scale it with high availability.
* **Serverless Architecture**: With Lambda, you don't need to worry about the underlying servers. AWS manages the infrastructure, including server maintenance and scaling, allowing you to focus on writing code.
* **Scalability**: Lambda automatically scales your application by running code in response to each trigger. Your code can scale from a few requests per day to thousands per second without any additional effort from you.
* **Billing**: You are billed based on the compute time your code consumes. This means you only pay for the time your code is running, making it a cost-effective solution for many applications.
* **Supported Languages**: Lambda supports multiple programming languages, including Node.js, Java, C#, Go, and Python. You can write your code in any of these languages and deploy it as a Lambda function.
* **Use Cases**: Lambda is ideal for scenarios where you need extra compute power for intensive tasks, want to decouple application components, or need to offload tasks from your local servers to the cloud.

**API Gateway**

* **API Gateway**: A managed service that allows you to create, publish, maintain, monitor, and secure APIs at any scale. It enables you to manage APIs in the cloud without worrying about local storage or infrastructure.
* **Serverless Architecture**: API Gateway works with AWS Lambda to create a serverless architecture, meaning you can run code without provisioning or managing servers. This allows for efficient data processing and function execution without the need for EC2 instances.
* **Integration with AWS Services**: API Gateway can interact with various AWS services like EC2, S3, and Lambda, as well as external web services. This flexibility allows you to build robust applications that can pull data from different sources.
* **Cross-Origin Resource Sharing (CORS)**: By default, API Gateway restricts requests to the same domain. Enabling CORS allows your API to handle requests from different domains, which is crucial for web applications that interact with multiple services.
* **Use Case Example**: Imagine a sales application where sales professionals input orders. The API Gateway sends this data to a Lambda function, which checks inventory in an RDS database and places orders if stock is low, all without managing any servers.

These concepts are essential for understanding how to effectively use API Gateway in your AWS environment.

Top of Form

Bottom of Form

**Kinesis**

* **AWS Kinesis**: A service for processing real-time streaming data. It allows you to analyze data as it flows in, rather than waiting for it to be stored in a database.
* **Kinesis Data Streams**: This component processes data in small chunks called shards. It can either store the data in a database like Redshift or analyze it on the fly.
* **Kinesis Data Firehose**: Combines data streaming and analytics, allowing you to process and analyze data before sending it to a consumer.
* **Kinesis Data Analytics**: Focuses on performing analytics on streaming data, providing real-time insights.
* **Kinesis Video Streams**: A newer service for processing video streams in real-time.
* **Benefits**: Fully managed architecture, no need for custom coding, and efficient real-time data processing. It is particularly useful for scenarios like social media sentiment analysis, where you need to analyze data as it comes in.

These concepts are essential for understanding how to use AWS Kinesis for real-time data processing and analytics.

**Kinesis Data Streams and Firehose**

* **Kinesis Data Streams**: This service ingests and stores data streams for processing. Data is stored in shards, which are small chunks of information. Processing tools like Kinesis Data Analytics, EC2 instances, or Lambda functions can pull data from these shards for analysis.
* **Kinesis Data Firehose**: Unlike Data Streams, Firehose continuously prepares and loads data to a specified destination as soon as it arrives. It acts like a firehose, dumping data directly into storage solutions like S3 buckets, Redshift data warehouses, Elasticsearch, or Splunk.
* **Kinesis Video Streams**: This service is designed for streaming video data. It allows you to connect surveillance cameras to the cloud, where the video data can be processed using machine learning tools like Amazon Rekognition to detect objects, people, and even audio keywords.

These concepts highlight the different ways AWS Kinesis can handle real-time data ingestion and processing, making it a powerful tool for various data streaming needs.

Top of Form

**Kinesis Data Analytics**

* **Kinesis Data Analytics**: This service allows you to analyze real-time data streams using standard SQL queries. It processes data as it flows in, enabling you to extract meaningful information on the fly.
* **Real-Time Data Analysis**: You can perform SQL queries on streaming data to filter and extract only the data you need. This helps in reducing the volume of data you have to process later.
* **Concurrent Consumers**: Kinesis Data Analytics supports multiple consumers simultaneously. This means different services like Redshift, S3, Elasticsearch, Lambda functions, or Kinesis Data Streams can pull data concurrently for various purposes.
* **Input Sources**: The data for Kinesis Data Analytics typically comes from Kinesis Data Firehose or Kinesis Data Streams. This data is then processed using SQL queries.
* **Output**: After processing, the data can be sent to various destinations like business analytics software, S3 buckets, or trigger alerts. This allows for further analysis or immediate action based on the processed data.
* **Use Case Example**: Imagine you have a stream of news articles. Kinesis Data Analytics can filter this stream to find articles relevant to your organization using SQL queries. The filtered articles can then be further analyzed using machine learning tools available in AWS.

These concepts are essential for understanding how to use Kinesis Data Analytics for real-time data processing and analysis.

**Reference architectures**

* **Reference Architectures**: These are predefined architecture plans provided by AWS for specific scenarios, such as regulatory compliance (e.g., HIPAA, PCI DSS). They offer design recommendations to ensure best practices.
* **Well-Architected Framework**: This framework includes best practices and design principles for building secure, high-performing, resilient, and efficient infrastructure on AWS.
* **Example Architecture**: The video shows an example architecture with components like AWS accounts, VPCs, availability zones, private subnets, DMZs, relational databases, S3 buckets, and various AWS services (e.g., AWS Config, CloudWatch, CloudTrail).
* **Quick Starts**: These are templates provided by AWS to quickly deploy reference architectures. You can find them at aws.amazon.com/quickstart and choose from categories like databases, machine learning, compliance, and more.
* **CloudFormation**: The video demonstrates how to use CloudFormation to implement these reference architectures by loading templates into the AWS Management Console.

These concepts help you understand how to use AWS's predefined architectures and tools to build efficient and compliant systems.

**Designing of Operational ExcellenceTop of FormBottom of Form**

**CloudFront**

* **CloudFront**: This is a content delivery network (CDN) service provided by AWS. It distributes content to various localized regions worldwide, reducing latency and improving data transfer speeds for end-users.
* **How It Works**: CloudFront caches data at regional edge caches and edge locations closer to the user. This reduces the time it takes to access data, improving performance.
* **Use Cases**:
  + **Static Website Content Delivery**: Accelerates the delivery of static content like images, stylesheets, and scripts.
  + **Streaming Video**: Enhances the performance of on-demand or live streaming video by caching it closer to the user.
  + **Security**: Supports encryption and secure delivery of content.
  + **Customization**: Allows for content customization at the edge, such as localization and translation.
* **Content Sources**: CloudFront can cache content from S3 buckets, media package channels, and HTTP servers.
* **Access Control**: You can set up public or restricted access to content, enforce HTTPS requirements, and apply geo-restrictions to control who can view the content.

These concepts highlight how CloudFront improves the performance and security of content delivery across the globe.

**Web application firewall (WAF)**

* **Web Application Firewall (WAF)**: A security solution that controls access to web application servers (HTTP/HTTPS). It can filter and monitor HTTP requests to protect against attacks.
* **Deployment**: The WAF is integrated within AWS and can be configured for instances, S3 buckets, and other HTTP/HTTPS content hosts. It doesn't require separate appliances or instances.
* **Access Control**: WAF can control access based on HTTP request details or source IP addresses. It can work with CloudFront and Elastic Load Balancers to filter requests before they reach the servers.
* **Configuration Options**:
  + **Allow all requests except specified ones**: Less restrictive, blocks only specified requests.
  + **Block all requests except specified ones**: More restrictive, allows only specified requests.
* **Monitoring**: WAF can be monitored using CloudWatch or CloudTrail, and alerts can be set up with SNS for specific events.
* **Error Handling**: If access is denied, WAF returns an HTTP 403 error (access denied). The default behavior can be set to allow or deny requests that don't match any rules.
* **Use Cases**: WAF is crucial for public-facing web applications to protect against external threats. For internal applications, it may be less critical but still useful for internal threat protection.
* **AWS Shield**: An additional service for enhanced threat detection and protection, especially against distributed denial of service (DDoS) attacks. Shield Standard is free, but Shield Advanced costs $3,000 per month.

Bottom of Form

**Simple Queue Service (SQS)**

* **Simple Queue Service (SQS)**: This is a messaging service provided by AWS that allows different parts of an application to communicate with each other by sending messages through a queue. This helps in decoupling the components, meaning they can operate independently.
* **Decoupling Applications**: By using SQS, you can break your application into smaller, manageable tasks that communicate through messages. This makes it easier to scale and maintain.
* **Message Producers and Consumers**: Producers are processes that send messages to the SQS queue, while consumers are processes that retrieve and process these messages.
* **Queue Types**:
  + **Standard Queue**: Offers high throughput and best-effort ordering, but messages might not be processed in the exact order they were sent.
  + **FIFO Queue**: Ensures messages are processed in the exact order they were sent but supports fewer transactions per second.
* **Redundancy and Scalability**: SQS automatically stores messages across multiple availability zones and scales to handle any number of messages.
* **Message Size and Retention**: Messages can be up to 256 KB and are retained for up to 14 days by default.

In your role, understanding how to implement SQS can help you design more efficient and scalable applications.

**Simple Notification Service (SNS)**

* **SNS vs. SQS**: SNS (Simple Notification Service) is different from SQS (Simple Queue Service). While SQS is about managing message queues, SNS is about sending notifications.
* **Publish-Subscribe Model**: SNS uses a "publish-subscribe" mechanism. Think of it like a magazine subscription. The publisher (magazine company) sends out magazines (messages) to all subscribers (people who subscribed to the magazine). In SNS, publishers send messages to topics, and subscribers receive messages from those topics.
* **Topics**: A topic is like a container for messages. You can name topics based on what kind of alerts they hold, such as "admin alerts" or "performance alerts".
* **Delivery Methods**: SNS can send notifications through various channels like HTTP/HTTPS, email, SMS (text messages), and even to other AWS services like Lambda or SQS.
* **Message Limits**: SNS messages can be up to 256 KB in size, but SMS messages have a smaller limit of 1600 bytes, so larger messages are split into multiple parts.
* **Use Case**: As an architect, you need to decide who needs to be notified and through which channel (text, email, etc.) to ensure they get the right message.

**Simple Workflow (SWF)**

* **Workflow Mapping**: This involves outlining the sequence of tasks or steps in a process to understand how work flows from start to finish. In AWS, this is done using Simple Workflow (SWF).
* **Simple Workflow (SWF)**: A tool in AWS that helps manage and coordinate tasks across different components of an application. It defines the sequence of events needed to complete a workflow.
* **Decoupled Applications**: SWF is used in decoupled applications to understand and manage the flow of tasks between different components, ensuring they work together without breaking the workflow.
* **Workflow Components**:
  + **Activities**: Tasks that need to be completed as part of the workflow.
  + **Decider Function**: Determines the best path or sequence of activities based on the outcomes of previous tasks.
  + **Domain**: A logical boundary within which the workflow operates, defined within SWF.
* **Activity Tasks and Workers**:
  + **Activity Tasks**: Specific instances of activities, like processing an order.
  + **Activity Workers**: Applications that perform the tasks within the workflow.
* **SWF Architecture**: Involves orchestration of workflows with a decider (an EC2 instance) that manages decision tasks and worker processes that execute the tasks.

Understanding these concepts can help you design and manage workflows in your software development projects more efficiently.

**Step Functions**

* **Step Functions**: AWS Step Functions are a replacement for Simple Workflow (SWF) and are recommended for new deployments. They use state machines to manage workflows.
* **State Machines**: Unlike SWF, which uses a decider, Step Functions use state machines to determine the flow of tasks. This involves defining states and transitions between them.
* **Tasks**: A task is a single unit of work within a state machine. For example, processing an image or retrieving metadata.
* **Choice**: This provides branching logic within the workflow. Based on conditions, the workflow can take different paths. For instance, if an image is of a supported type, it is processed further; if not, the workflow ends.
* **Parallel Processing**: Step Functions allow multiple tasks to run concurrently. For example, processing an image for recognition and generating a thumbnail at the same time.
* **Advantages**: Step Functions offer better management of workflows with parallel processing and state machines, making them more efficient and scalable.

These concepts should help you understand how Step Functions work and their benefits over SWF.

Top of Form

**Designing for Elasticity and Scalability**

**Bottom of Form**

**OpsWorks**

* **OpsWorks**: A configuration management service in AWS that allows you to define and deploy configurations using code. It automates the deployment, configuration, and management of applications.
* **Components**:
  + **OpsWorks Stacks**: The original model, where you create layers representing different components (e.g., MySQL, Red Hat Linux) and combine them into a stack.
  + **OpsWorks Chef Automate**: Uses cookbooks and recipes to define configurations. Recipes are equivalent to layers in Stacks.
  + **OpsWorks Puppet**: Utilizes a puppet master server with modules, which are similar to layers.
* **Use Cases**:
  + **Automated Deployment**: Ideal for deploying specific server configurations and applications quickly and efficiently.
  + **Operations**: Supports application and infrastructure updates.
  + **On-Premises and Cloud**: Can be used for both cloud and on-premises deployments.
* **Pre-Built Layers**: Includes support for programming environments like Ruby, PHP, Node.js, and Java, as well as AWS services like Amazon RDS and MySQL.

Understanding these concepts will help you leverage OpsWorks for efficient application deployment and management. Top of Form

Bottom of Form

**Cognito**

* **Cognito**: AWS Cognito is a service that provides user identity and data synchronization, commonly known as Single Sign-On (SSO).
* **Single Sign-On (SSO)**: Allows users to log in using existing identities from providers like Google, Facebook, or your local Active Directory network, without needing separate AWS accounts.
* **Identity Providers**:
  + **Public**: Supports Google, Facebook, and Amazon.
  + **Private**: Supports Active Directory with SAML (Security Assertion Markup Language).
* **Open Standards**: Cognito uses open standards like OAuth 2.0, SAML 2.0, and OpenID Connect for identity management.
* **Profile Management**: Manages user profiles without needing individual IAM accounts, and can scale to millions of users.
* **Use Cases**: Ideal for applications like mobile apps or games that require scalable user authentication and profile storage.
* **AWS Integration**: Controls access to AWS resources by defining roles and mapping users to those roles, enabling social logins and federated identity access.

These concepts should help you understand how Cognito works and its benefits.

**Elastic MapReduce (EMR)**

* **Elastic MapReduce (EMR)**: A service in AWS for processing large amounts of data quickly and efficiently using distributed computing.
* **Distributed Processing**: EMR uses a managed Hadoop framework to distribute processing tasks across a cluster of servers, reducing the workload on a single processor.
* **Data Sources and Processing**: Data is pulled from S3 buckets and processed using EC2 instances. You define the number of clusters needed for the job.
* **Cluster Nodes**:
  + **Master Node**: Coordinates job distribution across other nodes.
  + **Core Nodes**: Run tasks and store data.
  + **Task Nodes**: Only run tasks without storing data.
* **Use Case**: Ideal for large-scale data processing tasks, such as analyzing a file with millions of records. By distributing the workload across multiple nodes, tasks that would take days on a single machine can be completed in minutes or hours.

These concepts should help you understand how EMR works and its benefits.

**CloudFormation**

* **AWS CloudFormation**: A service that allows you to model and set up your Amazon Web Services resources using templates.
* **Templates**: These are JSON or YAML formatted text files that define the AWS resources you want to create and configure.
* **Stacks**: When you use a template to create resources, the collection of those resources is called a stack. Stacks help manage related resources as a single unit.
* **Change Sets**: These are used to update your stacks. They allow you to see what changes will be made before actually applying them.
* **Benefits**:
  + **Rapid Deployment**: Quickly deploy resources using predefined templates.
  + **Consistency**: Ensure consistent configurations across multiple environments.
  + **Reusability**: Use templates created by others or reuse your own for different projects.
* **CloudFormer**: A tool within CloudFormation that can create a template from your existing AWS resources, making it easier to replicate or modify your infrastructure.

These concepts should help you understand how CloudFormation works and its benefits.

Top of Form

**CloudFormation properties**

* **CloudFormation Properties**: These are specific attributes that you define in your CloudFormation templates to configure AWS resources.
* **Dynamic Nature**: AWS frequently updates CloudFormation by adding new properties and supporting more services. It's important to stay updated with these changes to utilize the latest features.
* **Defining Property Values**: When creating CloudFormation templates, you specify property values for the resources you are deploying. These properties determine the configuration and behavior of your AWS resources.
* **Requesting New Properties**: If a needed property is not available, you can contact AWS customer service to request its addition. AWS may add it in the future, making it available for everyone.
* **Example Announcement**: The video mentions an announcement from June 2019, where AWS added new properties for services like EC2, ECS, EFS, and S3. This highlights the continuous improvement and expansion of CloudFormation capabilities.

Understanding these concepts will help you effectively use CloudFormation for automating and managing your AWS infrastructure.

**CloudWatch**

* **CloudWatch Overview**: A monitoring service in AWS that keeps an eye on both cloud and on-premises systems 24/7.
* **Dashboards**: You can create custom dashboards to monitor different aspects of your systems, such as billing, performance, and security.
* **Logs and Events**:
  + **Logs**: CloudWatch collects logs from your systems, which you can use to build dashboards and monitor performance.
  + **Events**: These are specific occurrences within your logs that CloudWatch can track. When an event happens, it can trigger alarms or other actions.
* **Alarms**: CloudWatch can set alarms based on specific events. These alarms can notify you via SNS (Simple Notification Service) or take automated actions like launching or restarting instances.
* **On-Premises Monitoring**: CloudWatch can also monitor on-premises systems by pushing their logs to the cloud.
* **Use Cases**: Ideal for ensuring the operational health of your systems, receiving performance and security notifications, and automating responses to specific events.

These concepts should help you understand how CloudWatch works and its benefits.

**Trusted Advisor**

* **Trusted Advisor**: A tool in AWS that provides recommendations to help you optimize your AWS environment.
* **Key Areas**:
  + **Security**: Identifies security gaps and suggests improvements, such as enabling multifactor authentication (MFA) for the root account.
  + **Cost Optimization**: Recommends ways to reduce costs, like identifying underutilized resources.
  + **Performance**: Offers tips to enhance performance, such as optimizing your configurations.
  + **Fault Tolerance**: Suggests ways to improve the reliability of your applications.
  + **Service Limits**: Alerts you when you are approaching AWS service limits.
* **Dashboard Indicators**:
  + **Red Circle with Exclamation Mark**: Immediate action needed.
  + **Yellow Triangle with Exclamation Mark**: Consider taking action.
  + **Green Square with Check Mark**: Everything is good.
* **Actionable Insights**: Provides specific details about issues and recommendations, allowing you to download the results for offline analysis.

**Organizations**

* **Multiple AWS Accounts**: Companies can have multiple AWS accounts for various reasons, such as security boundaries or reaching account limits.
* **AWS Organizations**: This service allows you to manage multiple AWS accounts under a single umbrella, providing a centralized management point.
* **Centralized Management**: AWS Organizations offers a single interface for managing billing, account management, and permissions across all accounts.
* **Organizational Units (OUs)**: These are containers within AWS Organizations that help you structure your accounts hierarchically (e.g., by department like IT, accounting, etc.).
* **Policies**: You can attach policies to OUs to manage permissions at a higher level, which can be inherited by sub-OUs.
* **Cost Management**: AWS Organizations helps in managing costs effectively by providing a centralized view and control over all accounts.

These concepts help streamline the management of multiple AWS accounts, making it easier to handle security, billing, and organizational structure.

Top of Form

Bottom of Form

Top of Form

Bottom of Form

Top of Form

Bottom of Form

Top of Form

Bottom of Form

Bottom of Form

Top of Form

Bottom of Form

Top of Form

Bottom of Form

Top of Form

Bottom of Form

Top of Form

Bottom of Form

Top of Form

Bottom of Form

Top of Form

Bottom of Form