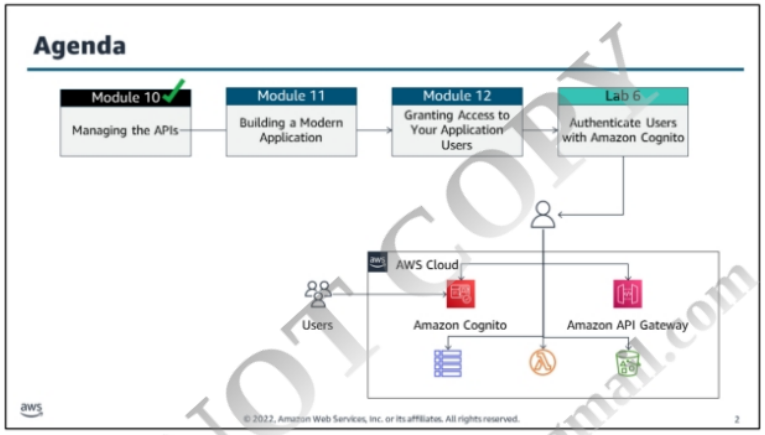
Printed by: sameersheik68@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.



Module 10 – Configure Amazon API Gateway as a way to connect all of the application's services together.

Module 11 – Evaluate the benefits of building a web application with a serverless approach.

Module 12 – Review how Amazon Cognito controls user access to AWS resources.

Lab 6 – Create an Amazon Cognito solution that provides users access to a web application.

Day 1 and 2 recap

Day 1

- Your application
- Developer tools (IDEs, SDKs, APIs)
- Permissions
- Storing and hosting your application

Day 2

- Databases
- Compute and storage
- API management

 3

Printed by: sameersheik68@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

Module objectives

By the end of this module, you will be able to:

- Describe the challenges with traditional architectures
- Describe the microservice architecture and benefits
- Explain various approaches for designing microservice applications
- Explain steps involved in decoupling monolithic applications
- Explain how to orchestrate Lambda functions using AWS Step Functions

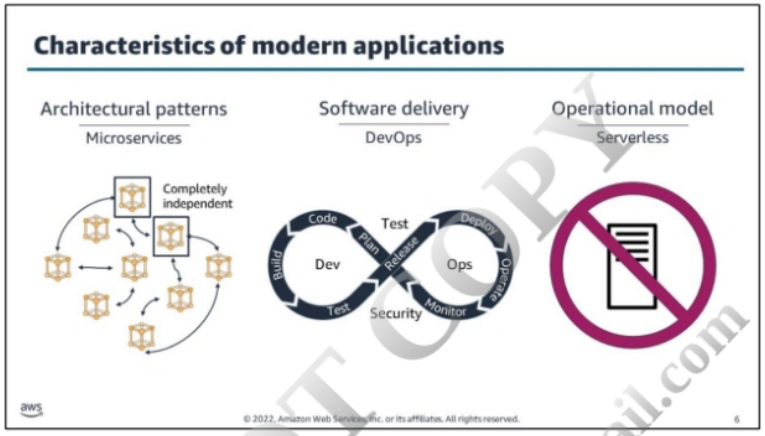
 4

You have been building a modern application. In this module, you learn what a modern application is and how it is developed. With a focus on microservices, you learn about the benefits of a developing a modern application and additional considerations such as their orchestration.

Other modules will focus on the development agility and observability of a modern application.

Printed by: sameersheik68@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

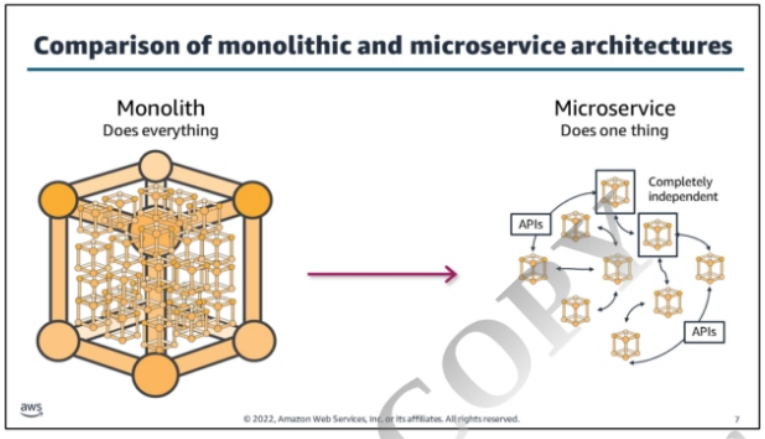
AWS Training and Certification Module 11: Building a Modern Application



AWS defines a modern application as a cloud-native application that is developed by a combination of modern technologies, architectures, software delivery practices, and operational processes. Modern applications are built with microservices architecture patterns, serverless operational models, and automated software delivery processes. By building modern applications, teams can deliver value more quickly, frequently, consistently, and safely.

A modern application is developed with a focus on the following characteristics:

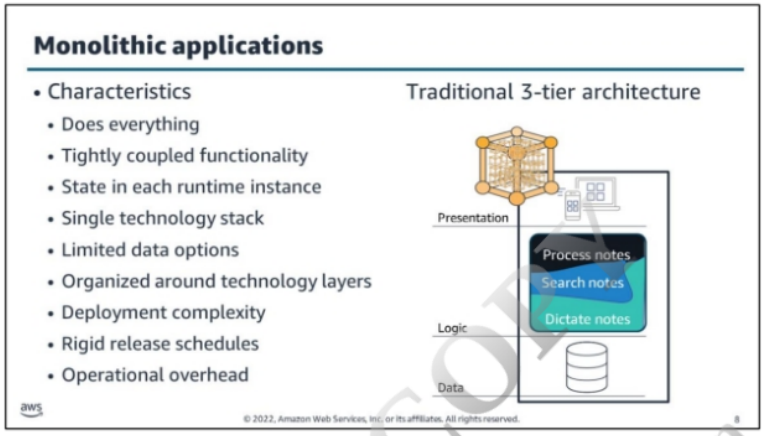
- **Speed** – Can be updated quickly based on customer and market needs.
- **Scale** – Scales quickly to potentially millions of users, can handle petabytes of data, and is globally available.
- **Resilience** – Resilient in the lifecycle and while deployed. Resiliency is built in the application to allow it to react quickly and recover from failure.
- **Security** – Secure at the application and the lifecycle levels because security is set for each service.



What is a monolithic application?
A *monolith* application has tightly coupled services in which the user interface and data are combined into a single program from a single platform. It is self-contained and independent from other computing applications.

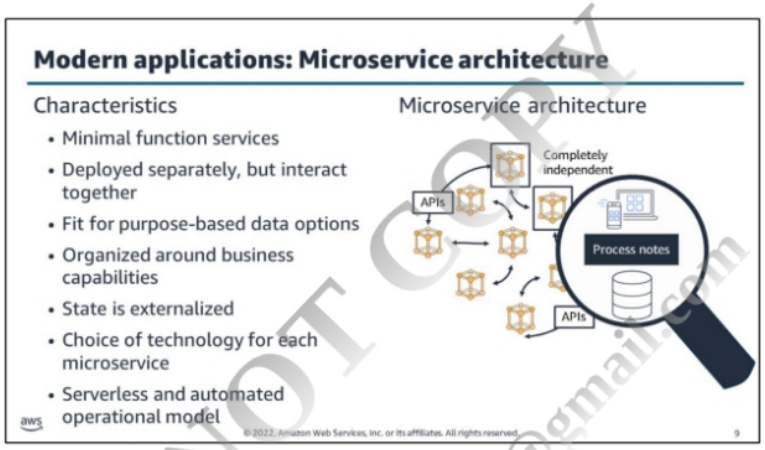
What is a microservice application?
A *microservice* application service has the following characteristics:
- Tightly scoped
- Strongly encapsulated
- Loosely coupled
- Independently deployable
- Scalable

Printed by: sameersheik68@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.



A traditional 3-tier architecture is layered and logically modular, consisting of the following components:
- **Presentation –** This layer manages HTTP requests and responds with HTML or JSON/XML (web services APIs).
- **Business logic –** Application servers
- **Data layer –** This layer includes the data access objects that access the database servers.

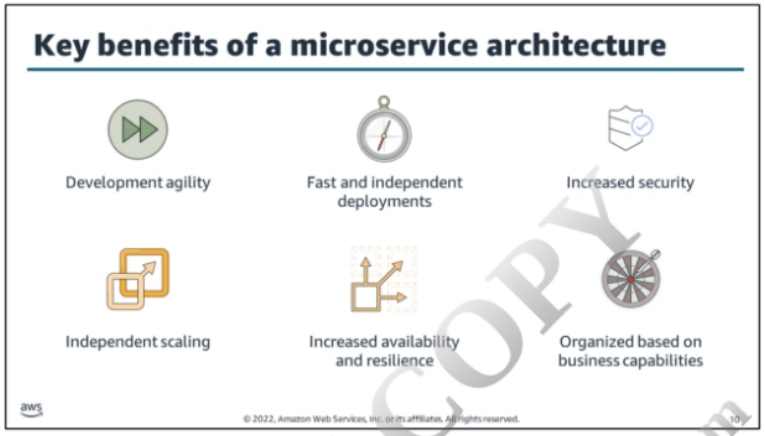
Drawbacks
Though this architecture is functional, if your business is fast-moving and fast-growing, you are not able to start or update an application as quickly as you want to. For example, the monolithic architecture presents the following challenges or drawbacks:
- You can deploy only at the pace of the slowest component within the monolith.
- If the coupling is high, so are the dependencies. As a result, your development and test complexity increases.



A modern application consists of many of the same components you have seen in the monolithic application, such as the data, logic, and presentation layers. But there are also a few key differences. Each microservice has a clean separation from the other microservices, each having its own persistence mechanisms.

Creating microservices or refactoring an existing monolith can be difficult.

Printed by: sameersheik68@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.



Because microservices are small and independent units, they provide the following benefits:

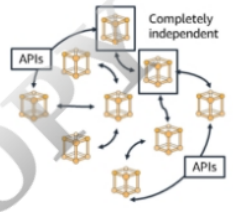
- **Development agility**
  - Code is readable.
  - Technology stack is easier to change.
  - Collaboration increases in small teams.
  - Changes are made independently of other services
- **Fast and independent deployments**
  - Microservices are deployed independently.
  - Deployment times are shorter.
  - Deployment process is easier to automate.
- **Increased security**
  - Each service is secure.
- **Independent scaling**
  - A microservice can scale independently.
- **Increased availability and resilience**
  - Fault isolation is improved.
- **Organized based on business capabilities**
  - Reuse services in other applications.

Starting a new development effort

- Enable experimentation by creating a culture of ownership
- Componentize applications using microservices
  - Well-defined interfaces through APIs
  - Each microservice is optimized for a single function
  - Each service is independent of other services

Example: Domain-driven design

- Functionality of microservice is within the context of the domain
- Defines the integration points with other domains



 11

Printed by: sameersheik68@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

Interaction patterns

API-Driven

- Synchronous processing
- Applications and services connect and communicate through APIs
  - AWS services expose their APIs
- HTTP and HTTPS communication protocol

Event-Driven

- Asynchronous processing
- A message represents something has happened
  - AWS resources can generate events when their state changes
- Consumer subscribes to an event
- Provider generates an event

 12

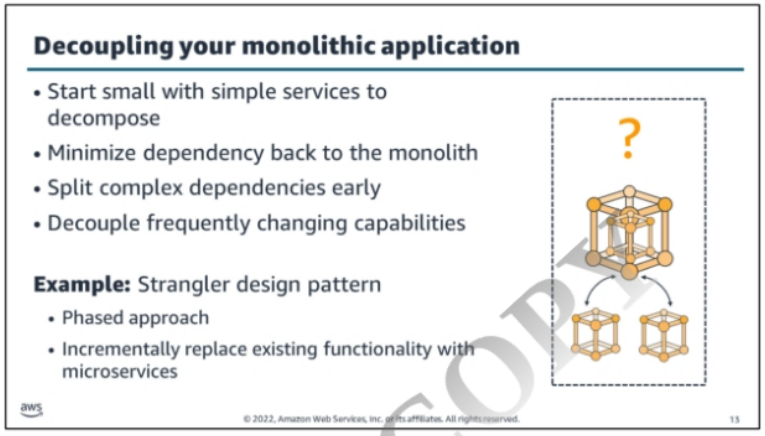
Integration falls into two categories:

- Synchronous or API-based
- Asynchronous or event-based

Considerations

Modern applications use both types of integrations. When deciding on who to integrate, consider the following:

- Are you communicating within a service or between services? In general, services are wrapped in an API but use messaging and events internally.
- Event-driven communication is nearly infinitely scalable, and also scales to 0, which results in a cost benefit.
- Event-driven applications enable the decoupling of services and of the teams that built them. This decoupling increases team agility.

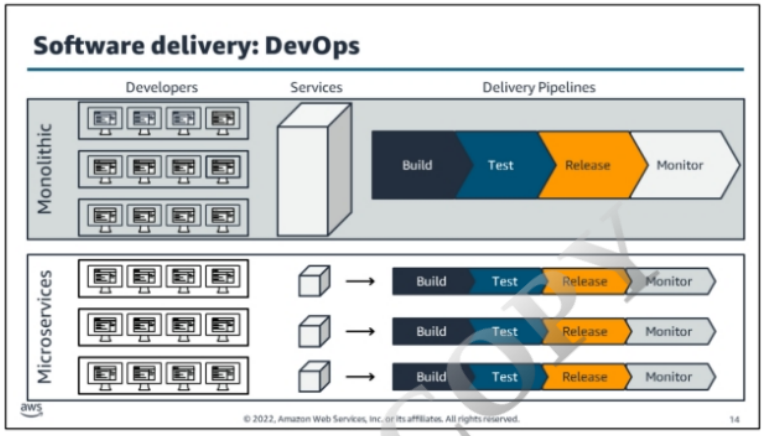


Decoupling a monolithic application requires a well-thought-out process, which is implemented gradually.

To learn more, see "Advanced Developing on AWS" on the AWS Training website (https://aws.amazon.com/training/classroom/advanced-developing-on-aws/).

Printed by: sameersheik68@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

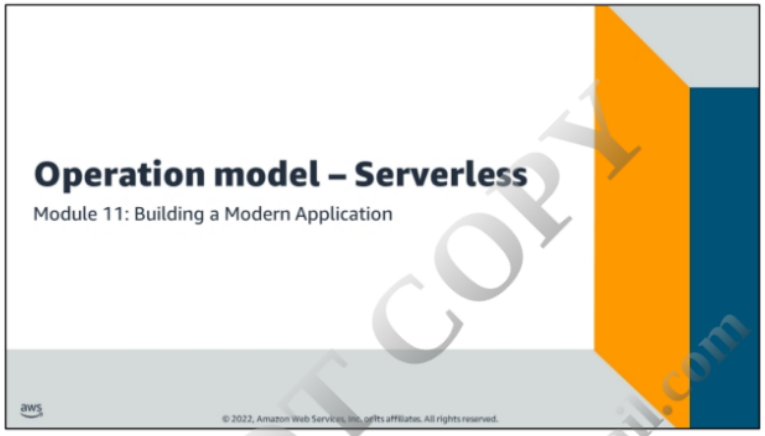
AWS Training and Certification Module 11: Building a Modern Application



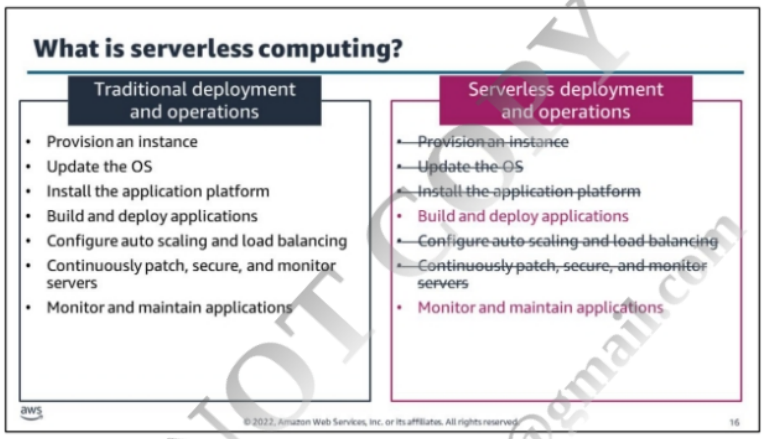
A traditional application has one release pipeline that is shared among many teams. This process often introduces bottlenecks. When teams have the authority to move fast, they can release new features continuously, often several times per day.

The way you operate an application that releases multiple times a day is different from one that releases once or twice a year. Under a DevOps model, development and operations teams take ownership of service with shared responsibility. Through a combination of cultural philosophies, practices, and tools, they find ways to deliver applications and services at high velocity.

© 2022 Amazon Web Services, Inc. or its affiliates. All rights reserved.

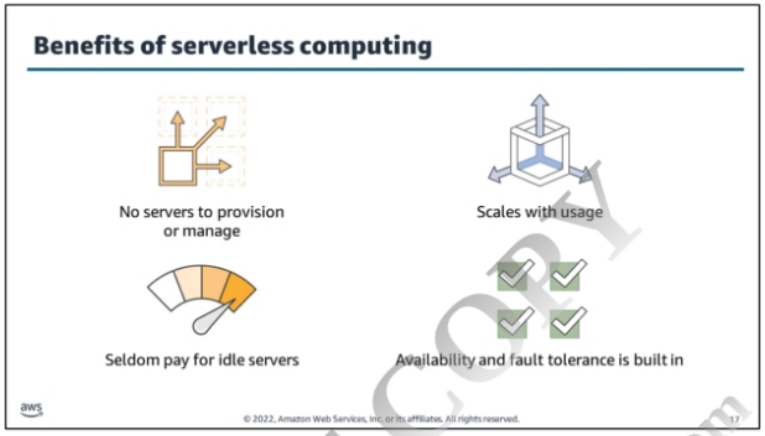


Printed by: sameersheik68@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.



With serverless computing, you can build and run applications and services without considering servers. Serverless applications don't require you to provision, scale, and manage any servers. You can build services for nearly any type of application or backend service. Everything required to run and scale your application with high availability is handled for you.

Building serverless applications means that you can focus on your core product. You don't have to worry about managing and operating servers or runtimes in the cloud or on premises. By reducing this overhead, as a developer, you can reclaim time and effort that can be spent on developing scalable and reliable products.



Benefits include the following:

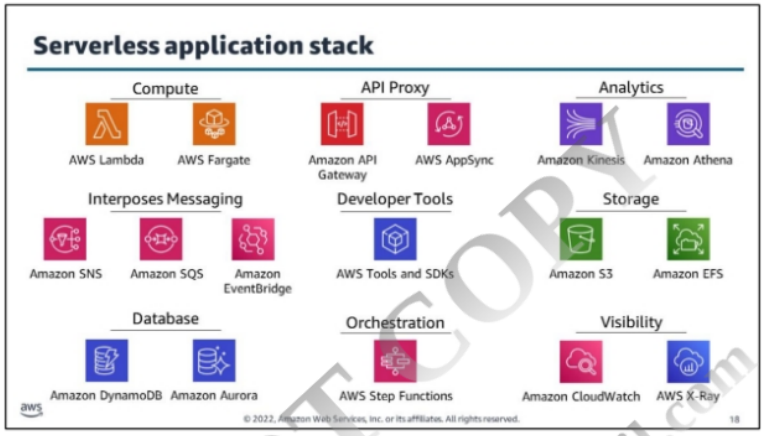
- Don't pay for idle, request-based pricing.
- No infrastructure to provision, monitor, or manage
- Scaling and fault-tolerance are built in
- Easier to author, deploy, and secure
- Enables best practices (events, stateless functions)

No need to provision or maintain any servers. No software or runtime is required to install, maintain, or administer.

You can automatically scale your application. Alternatively, you can scale the application by adjusting its capacity through toggling the units of consumption (throughput or memory) instead of units of individual servers.

Serverless applications have built-in availability and fault tolerance. You don't need to architect for these capabilities because the services running the application provide them by default.

Printed by: sameersheik68@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.



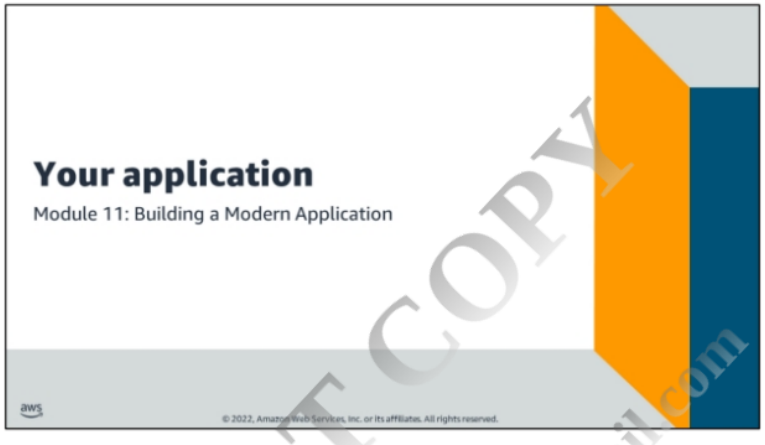
AWS provides a set of fully managed services that you can use to build and run modern applications.

AWS developed serverless services for all three layers of your stack, including:

- **Compute** – AWS Lambda and AWS Fargate
- **Application integration** – Amazon EventBridge, AWS Step Functions, Amazon Simple Queue Service (Amazon SQS), Amazon Simple Notification Service (Amazon SNS)
- **Data stores** –Amazon Simple Storage Service (Amazon S3), Amazon DynamoDB, Amazon Elastic File System (Amazon EFS), or Amazon Aurora

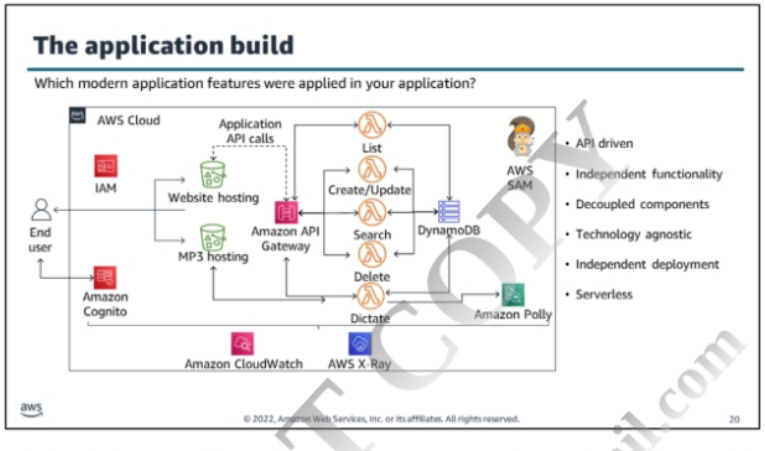
However, other services are available to support your application lifecycle after production, such as Amazon CloudWatch or AWS X-Ray.

For more information, see "Serverless on AWS" on the AWS Product website (https://aws.amazon.com/serverless/).



Printed by: sameersheik68@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

AWS Training and Certification Module 11: Building a Modern Application

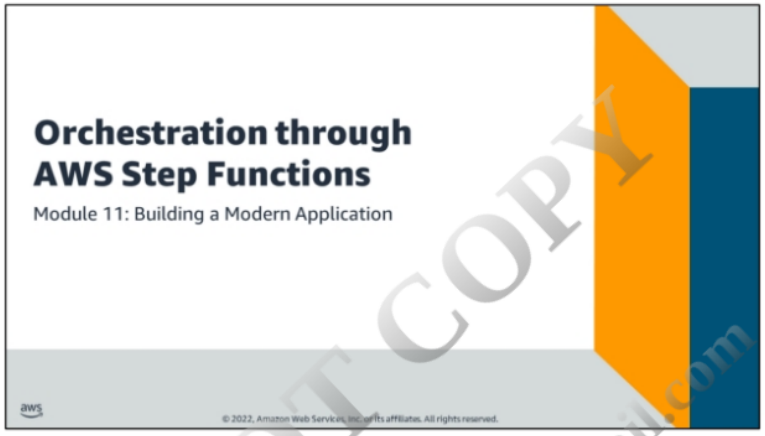


Consider the application you have built in this course. This is a cloud-native, serverless, modern application.

What you have built so far by applying serverless concepts:
- API driven
- Independent functionality
- Decoupled components
- Technology agnostic
- Independent deployment
- Serverless

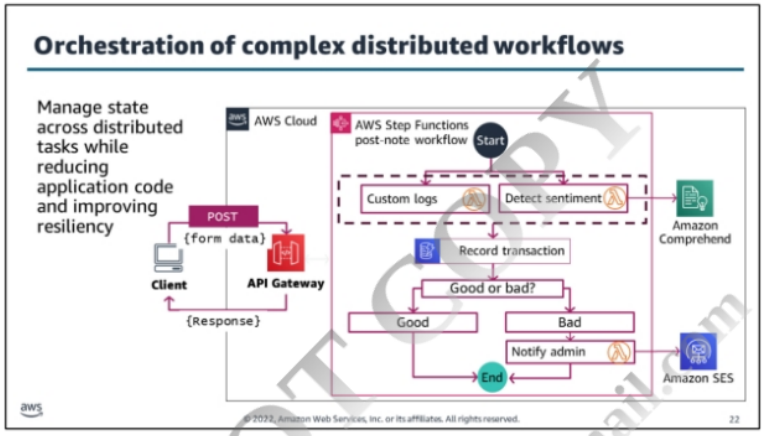
Missing components because it's serverless:
- Load balancers
- Servers

What else can be done?



Printed by: sameersheik68@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

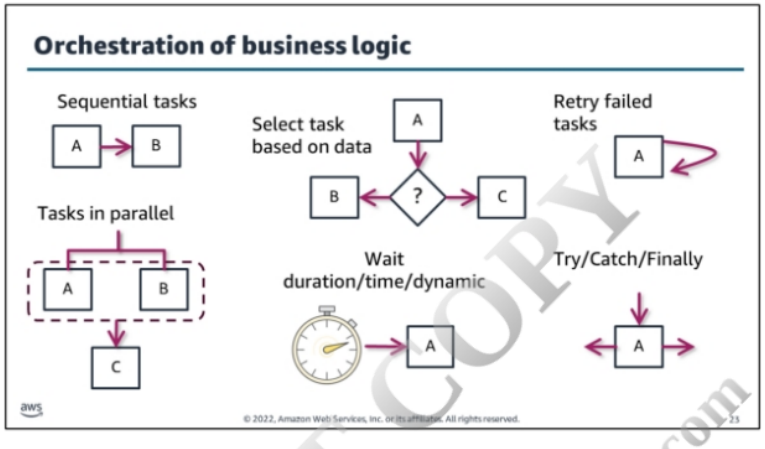
AWS Training and Certification Module 11: Building a Modern Application



In general, modern cloud applications are composed of many services and components. As applications grow, an increasing amount of code is built to coordinate the interaction of all the components.

In this example, you see a workflow for a Step Function for Post note. Imagine a richer functionality to the one you have used in your application within the course.
What if before posting a note, you want to record the request in a custom log and detect its sentiment using Amazon Comprehend? Recording a log entry and detecting the sentiment can happen in parallel, and when complete, the transaction can be recorded. **Good** or **bad** can be a number of things as defined by the workflow. It can be an error, or even flagged inappropriate language.

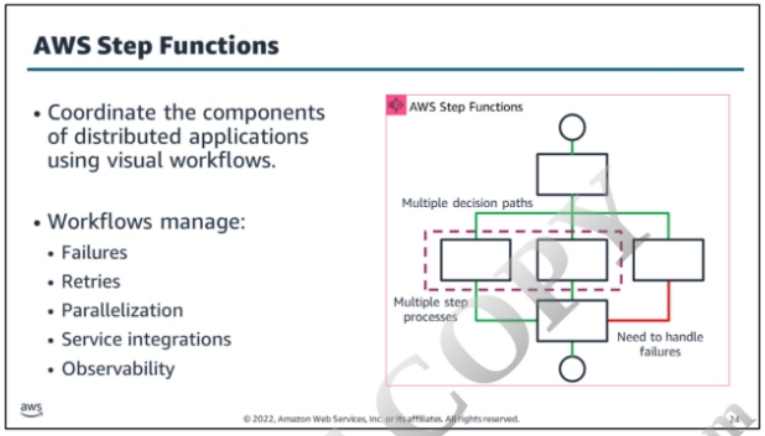
AWS Step Functions provides serverless orchestration of such interactions.



Serverless applications can become complex. Your business logic has functions or operations for which you need to orchestrate your code. Instead of writing code through your Lambda functions, which handle business logic, retries, and time outs, you can use Step Functions.

Printed by: sameersheik68@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

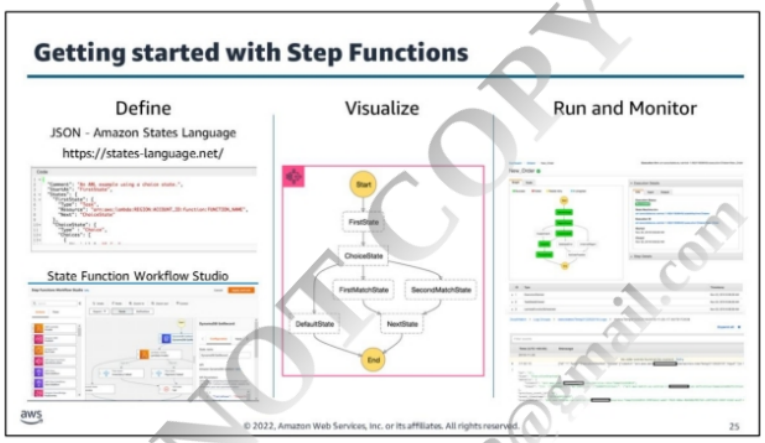
AWS Training and Certification Module 11: Building a Modern Application



AWS Step Functions is a serverless orchestration service that lets you combine AWS Lambda functions and other AWS services to build business-critical applications.

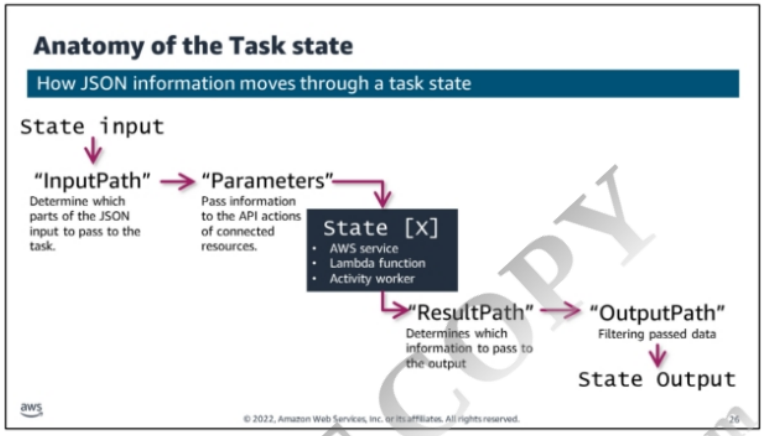
A step function is based on *state machines* and *tasks*.

- A *state machine* is a workflow with built-in controls.
- A *task* is a state in a workflow that represents a single unit of work that another AWS service performs.
- Each step in a workflow is a *state*. A state of each step is examined to make sure that the application runs in order and as expected.



The workflows of the AWS Step Functions can be defined using a JSON-based language called the Amazon States Language, or using the Workflow Studio. The state machine defined in the workflow can be visualized during design time. From the AWS Management Console, you can visualize the workflow while it is running. You can also use Amazon CloudWatch metrics, CloudWatch Logs, AWS X-Ray, and other tools to log and monitor the workflows.

Printed by: sameersheik68@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.



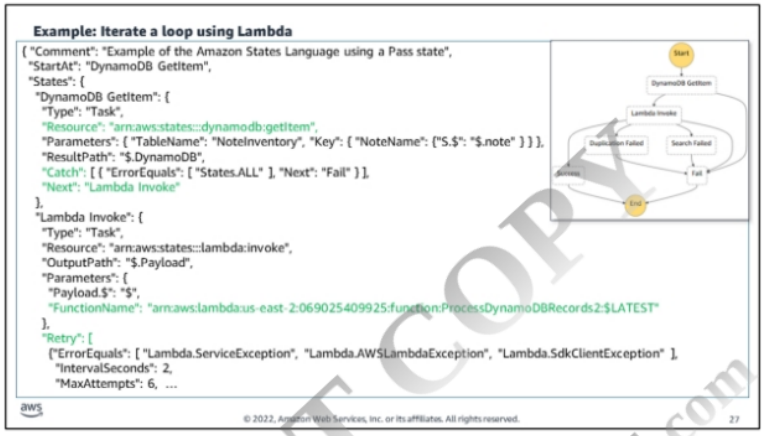
AWS Step Functions use workflows defined as state machines. Workflows specify how states communicate and how data is passed between them.

States can perform a variety of functions in your state machine, including:

- **Task** – Performs a single unit of work by doing the following:
- **Choice** – Choose the appropriate branch of the flow.
- **Fail or Succeed** – Stop the flow based on pass/fail.
- **Pass** – Pass the input to the output as is, or combined with some fixed data.
- **Wait** – Delay the flow for a specific amount of time.
- **Parallel** – Begin parallel branching within the flow.
- **Map** – Run the same steps for multiple entries of an array in the state input.

Individual states receive JSON as input and usually pass JSON as output to the next state. The InputPath, Parameters and ResultSelector fields provide a way to manipulate JSON as it moves through the workflow.

All work in a state machine is performed by tasks. The AWS Step Functions helps coordinate the actions between the various tasks. The input information goes into the `InputPath`. Its parameters are considered, and the state is set. The code is then invoked. The `ResultPath` is determined with the output data.



Step Functions is a serverless orchestration service that lets you combine AWS Lambda functions and other AWS services to build business-critical applications.

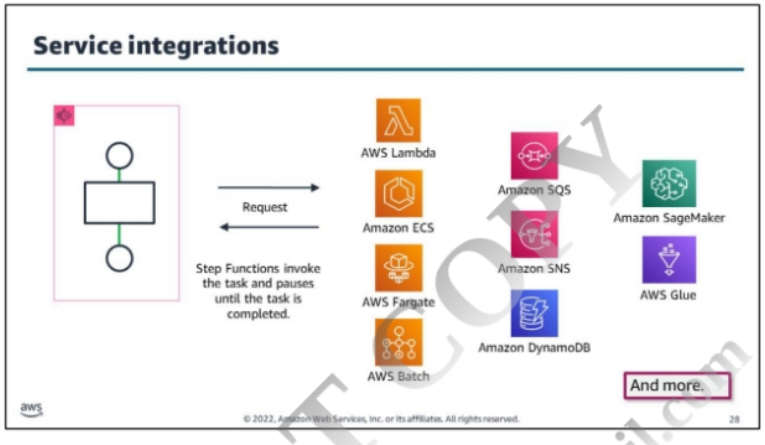
This example is a JSON representation of a state machine with several task states.

The state machine starts at state `DynamoDBGetItem`.

State `DynamoDBGetItem` is of type `Task`. This task is performed by the resource whose Amazon Resource Name (ARN) is used. The `tableName: NoteInventory` parameter and the key parameters use the "$" notation to pull the value from the corresponding input. The `ResultPath` is updated with the retrieved value of the item. If an error occurs, the error is caught, and the flow is sent to the `Fail` state. If it is successful, the flow continues to the state `Lambda Invoke`.

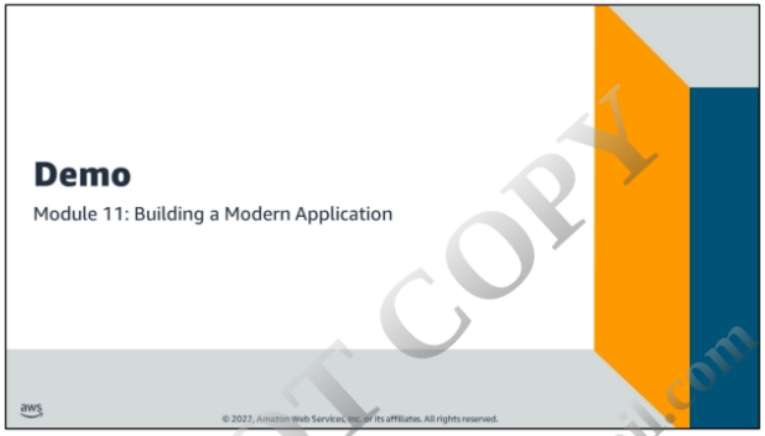
In the Lambda Invoke state, notice the Retry statement.

Printed by: sameersheik68@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.



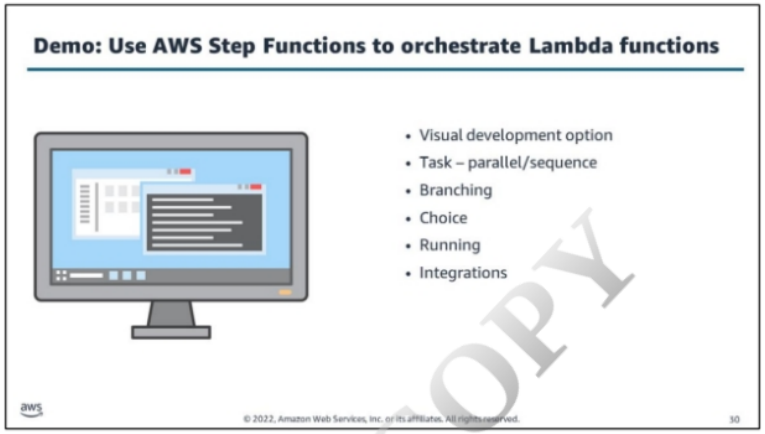
AWS Step Functions integrate with services using three service integration patterns:

- Call a service and let Step Functions progress to the next state immediately after it gets an HTTP response.
- Call a service and have Step Functions wait for a job to complete.
- Call a service with a task token and have Step Functions wait until that token is returned with a payload.

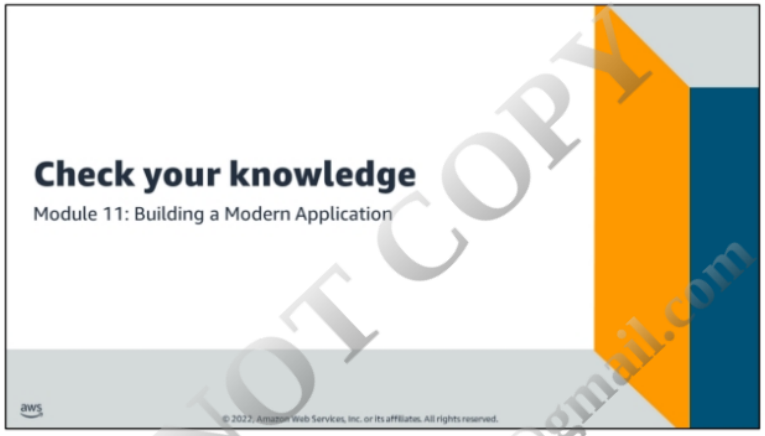


Printed by: sameersheik68@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

AWS Training and Certification Module 11: Building a Modern Application



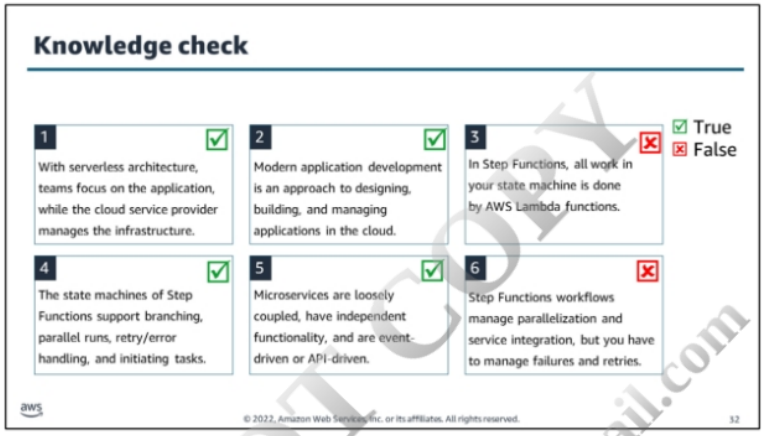
DO NOT COPY
sameersheik68@gmail.com



Printed by: sameersheik68@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

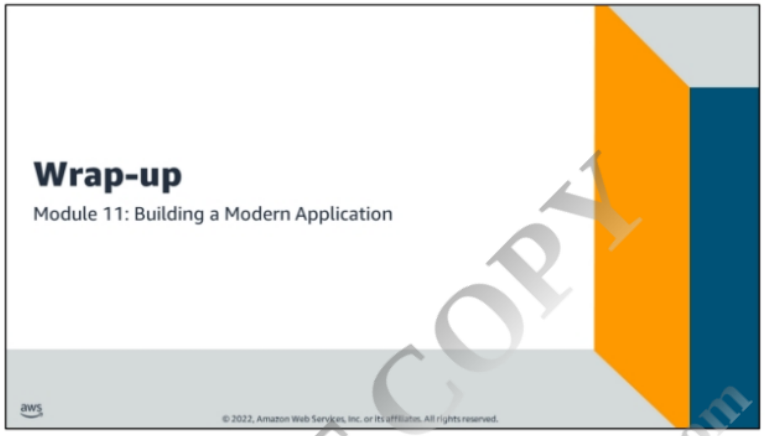
AWS Training and Certification

Module 11: Building a Modern Application



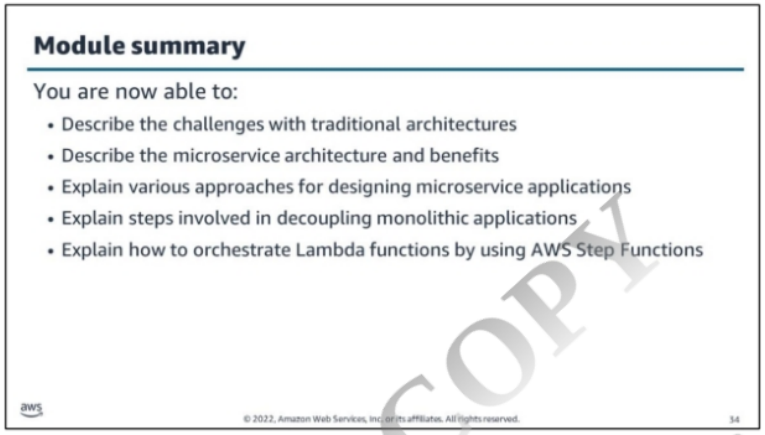
1. (True)
2. (True)
3. (False) All work in your state machine is done by ***tasks***, which can invoke Lambda functions or other services.
4. (True)
5. (True)
6. (False) Workflows manage failures, retries, parallelization, service integrations, and observability.

© 2022 Amazon Web Services, Inc. or its affiliates. All rights reserved.



Printed by: sameersheik68@gmail.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

AWS Training and Certification
Module 11: Building a Modern Application



DO NOT COPY
sameersheik68@gmail.com