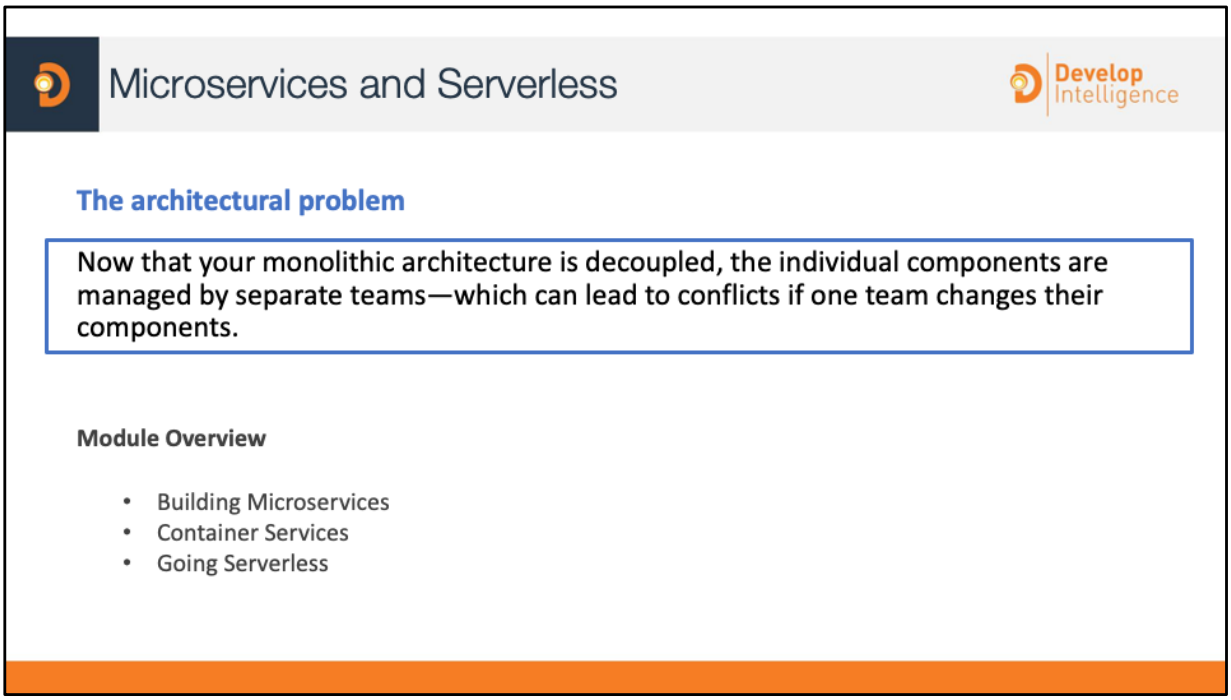
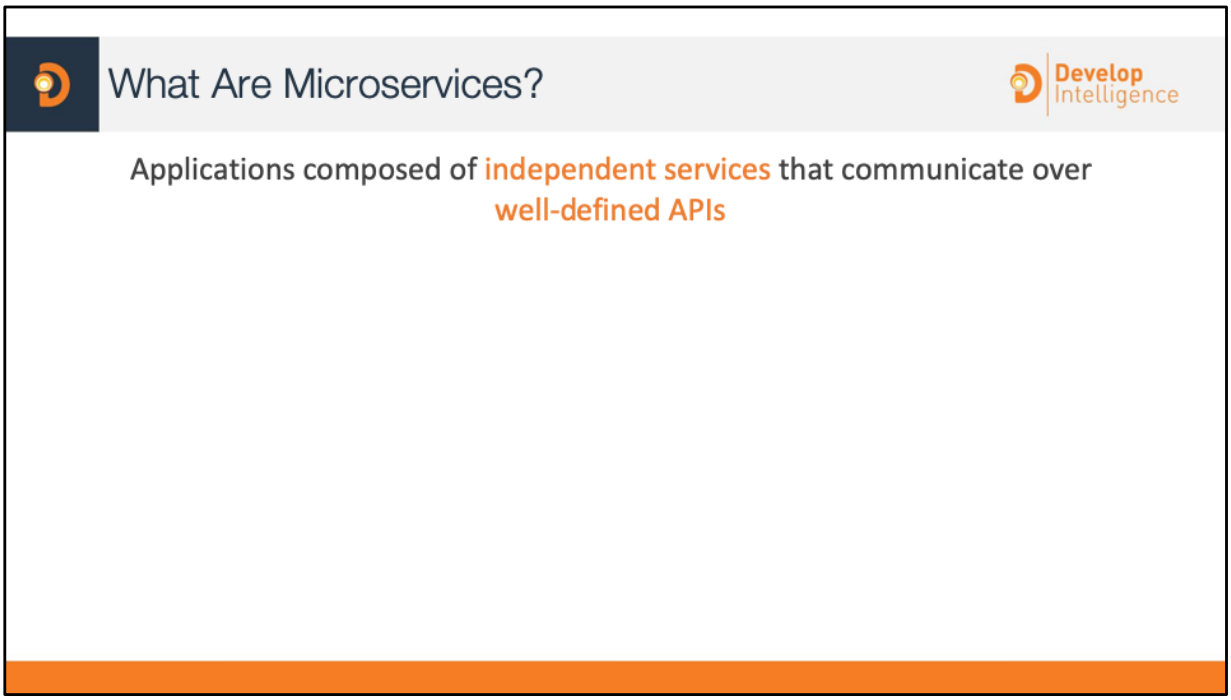
Microservices and Serverless Architectures

Develop Intelligence

- Module 12


Microservices and Serverless
Develop Intelligence
The architectural problem
Now that your monolithic architecture is decoupled, the individual components are managed by separate teams—which can lead to conflicts if one team changes their components.
Module Overview
• Building Microservices
• Container Services
• Going Serverless



The traditional monolithic application contains all the moving and working pieces, which are tightly bound together. If one piece were to fail, the entire application would crash. If there was a spike in demand, the entire architecture must be scaled. Adding features to a monolithic application becomes more complex as time wears on. Pieces of the codebase must interweave with each other to sync properly.

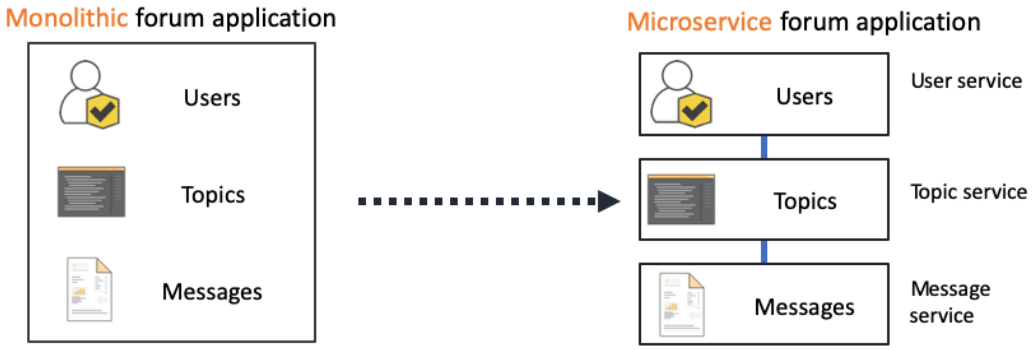
With a microservices architecture, an application is built as independent components that run each application process as a service. These services communicate via a well-defined interface using lightweight APIs. Services are built for business capabilities, and each service performs a single function. Because they are independently run, each service can be updated, deployed, and scaled to meet demand for specific functions of an application.

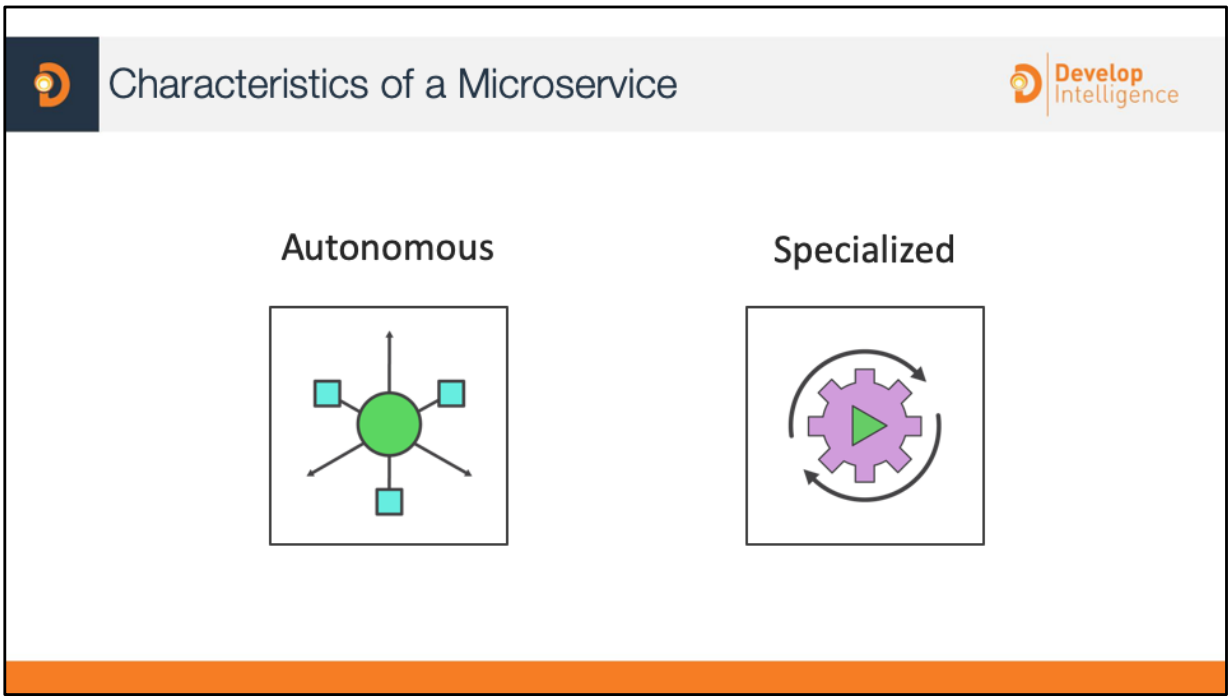
For an overview of microservices on AWS, see https://aws.amazon.com/microservices/

What Are Microservices?

Develop Intelligence

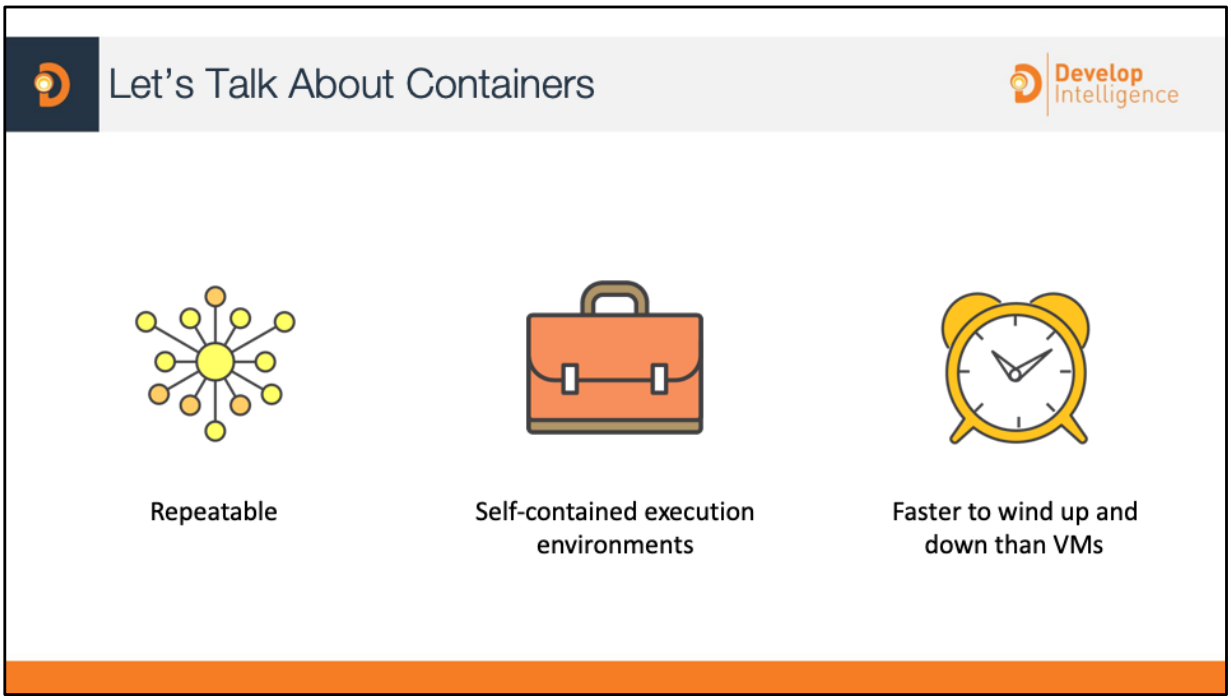
Applications composed of independent services that communicate over well-defined APIs



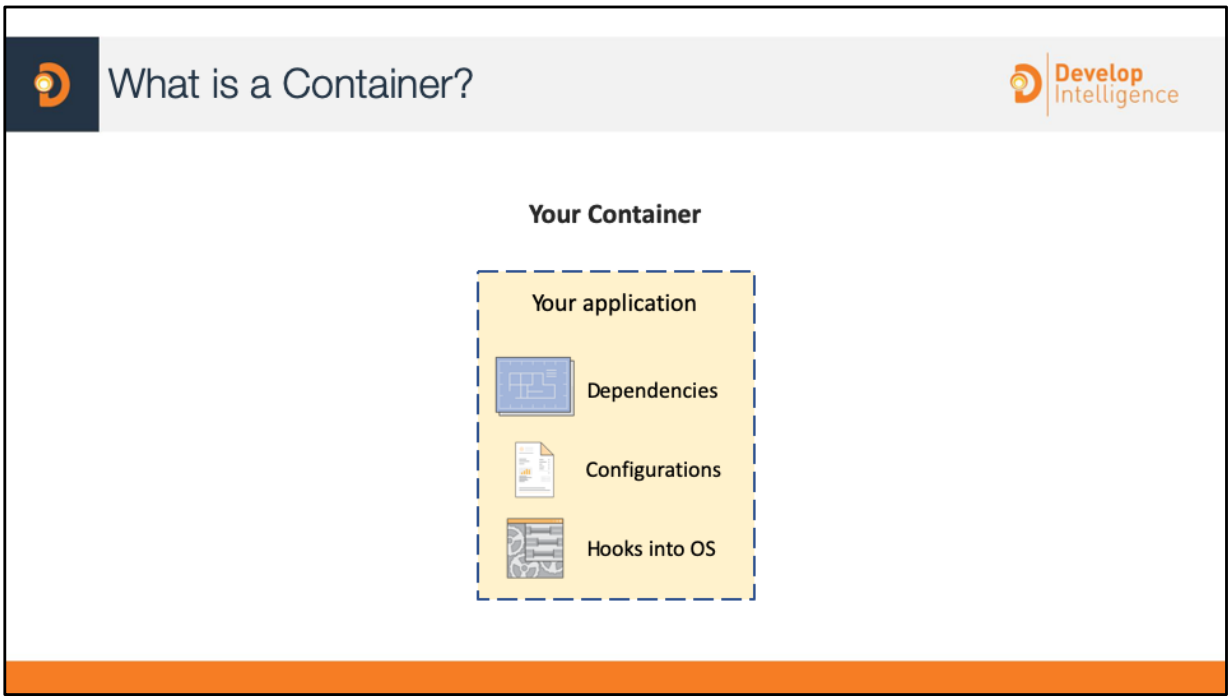


Autonomous
Each component service in a microservices architecture can be developed, deployed, operated, and scaled without affecting the functioning of other services. Services do not need to share any of their code or implementation with other services. Any communication between individual components happens via well-defined APIs.

Specialized
Each service is designed for a set of capabilities and focuses on solving a specific problem. If developers contribute more code to a service over time and the service becomes complex, it can be broken into smaller services.



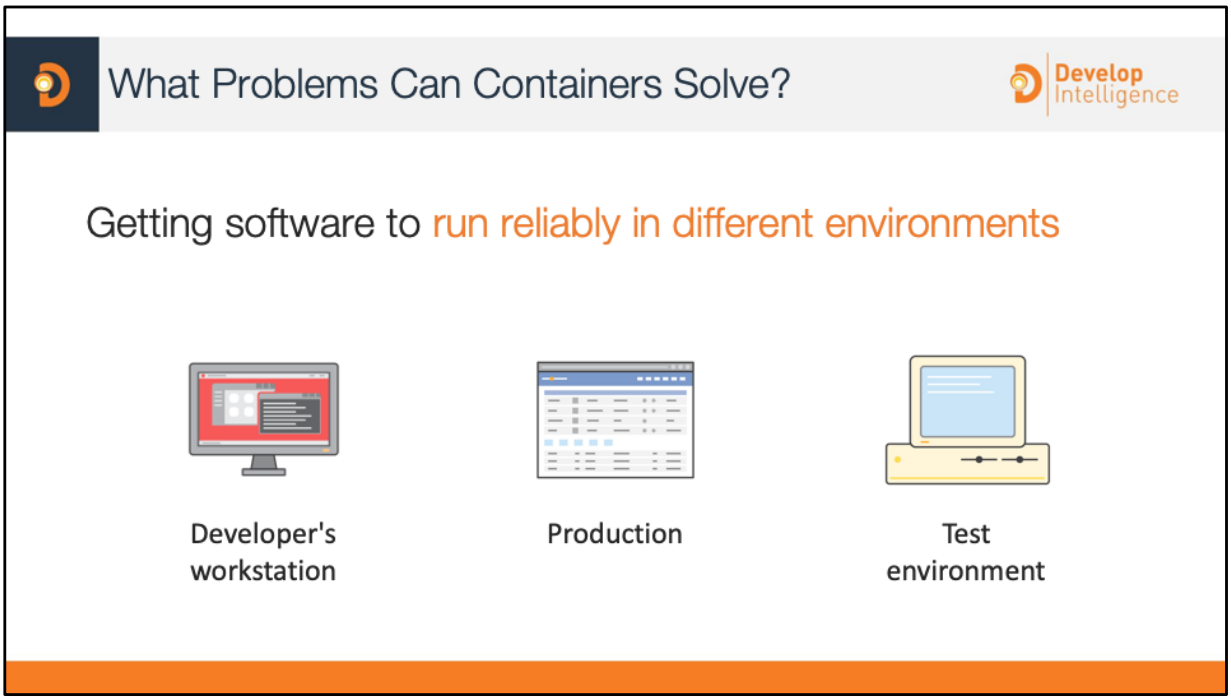
The benefits of a microservice-oriented architecture should trickle down also at an infrastructure level, where—up to this point of the course—you are still using virtual machines as your execution environment. While running VMs in the cloud gives you a dynamic, elastic environment, you may want to further reduce friction.



Containers are a method of operating system virtualization that allow you to run an application and its dependencies in resource-isolated processes. By using containers, you can easily package an application's code, configurations, and dependencies into easy-to-use building blocks that deliver environmental consistency, operational efficiency, developer productivity, and version control.

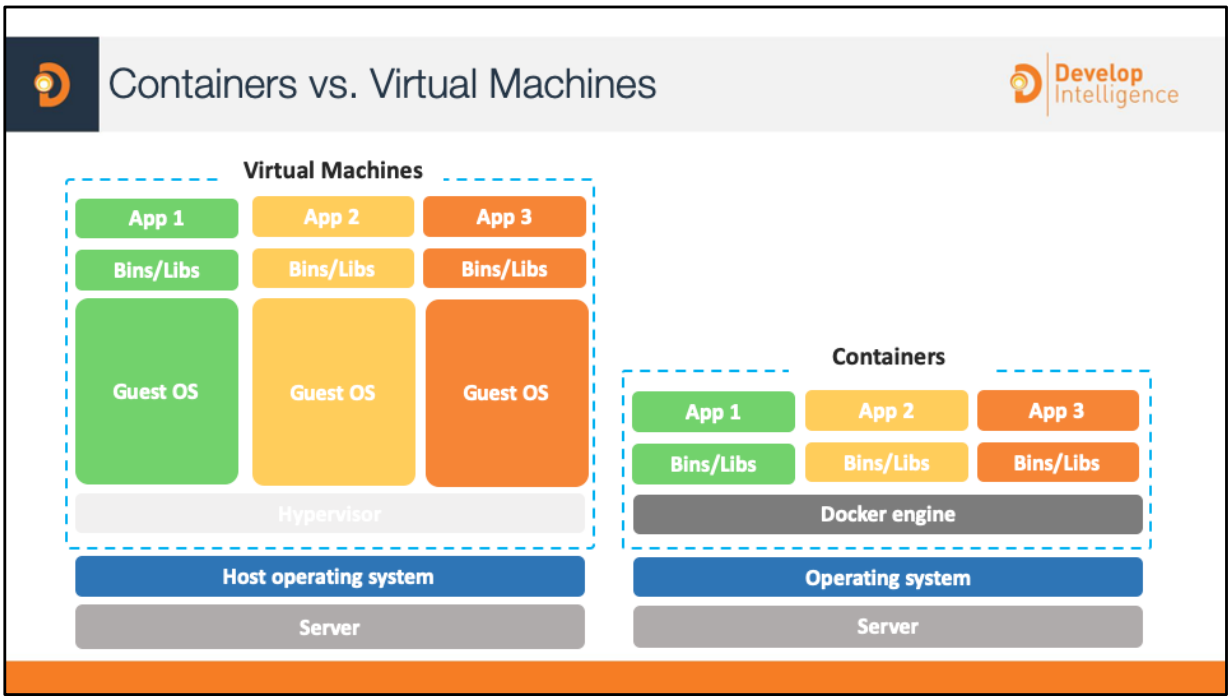
A container image is the snapshot of the file system available to the container. For example, you could have the Debian operating system as a container image: when running such container, you will effectively have a Debian operating system available in the container. You could also package all your code dependencies in the container image and use that as your code artifact. It is worth noting that container images are usually an order of magnitude smaller than virtual machines in terms of space. Spinning up a container is a matter of hundreds of milliseconds.

So by using containers, you can use a fast, portable, infrastructure agnostic execution environment.



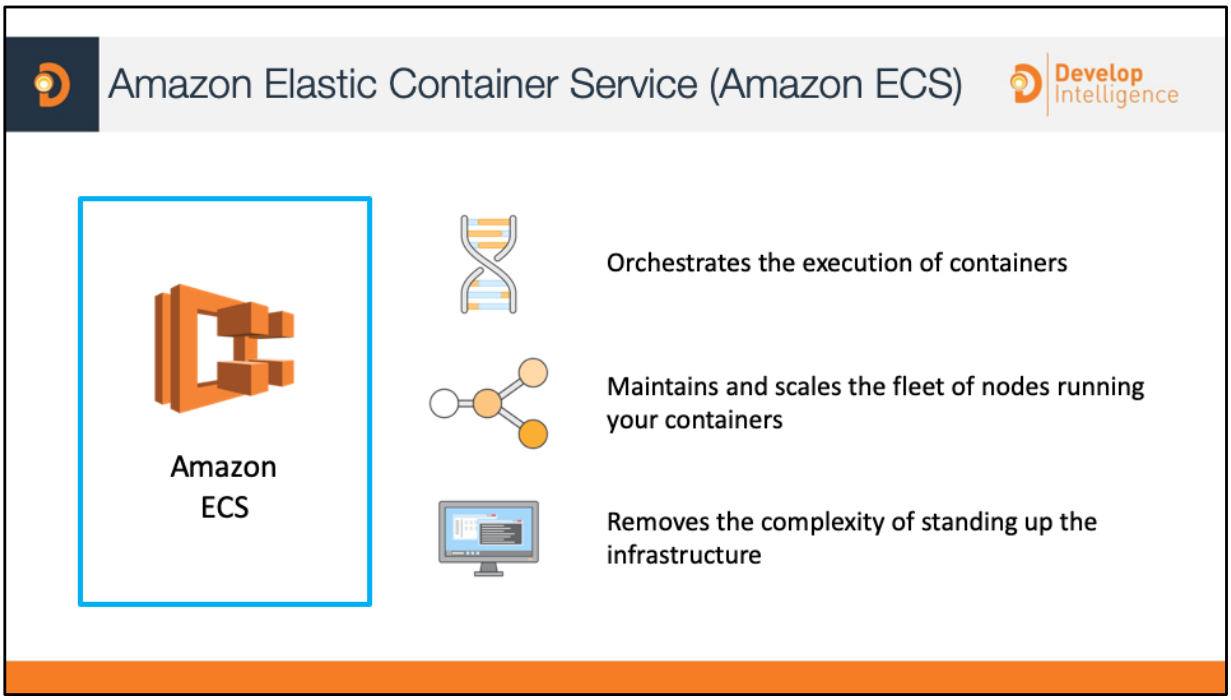
Containers can help ensure that applications deploy quickly, reliably, and consistently, regardless of deployment environment. Containers also give you more granular control over resources, which gives your infrastructure improved efficiency.

If you are looking for premade container solutions, you may want to visit the AWS Marketplace for Containers, which helps you to find and buy container products from independent software vendors through both the Amazon ECS console and AWS Marketplace. These verified and commercially-supported products run on Docker-compatible AWS container services such as Amazon ECS, AWS Fargate, and Amazon EKS. You can choose from product categories, such as high-performance computing, security, and developer tools. In addition, SaaS products that manage, analyze, or protect container applications are available. For more information, see https://aws.amazon.com/marketplace/features/containers.



When hearing the capabilities of containers, it is somewhat intuitive to think that it sounds just like a virtual machine. However, the differences are in the details. The number one difference is the lack of a hypervisor requirement. Containers can run on any Linux system with appropriate kernel feature support and the Docker daemon present. This makes them extremely portable. Your laptop, your VM, your EC2 instance, and your bare metal server are all potential hosts.

The lack of a hypervisor requirement also results in almost no noticeable performance overhead. The processes are talking directly to the kernel and are largely unaware of their container silo. Most containers boot in just a couple of seconds.

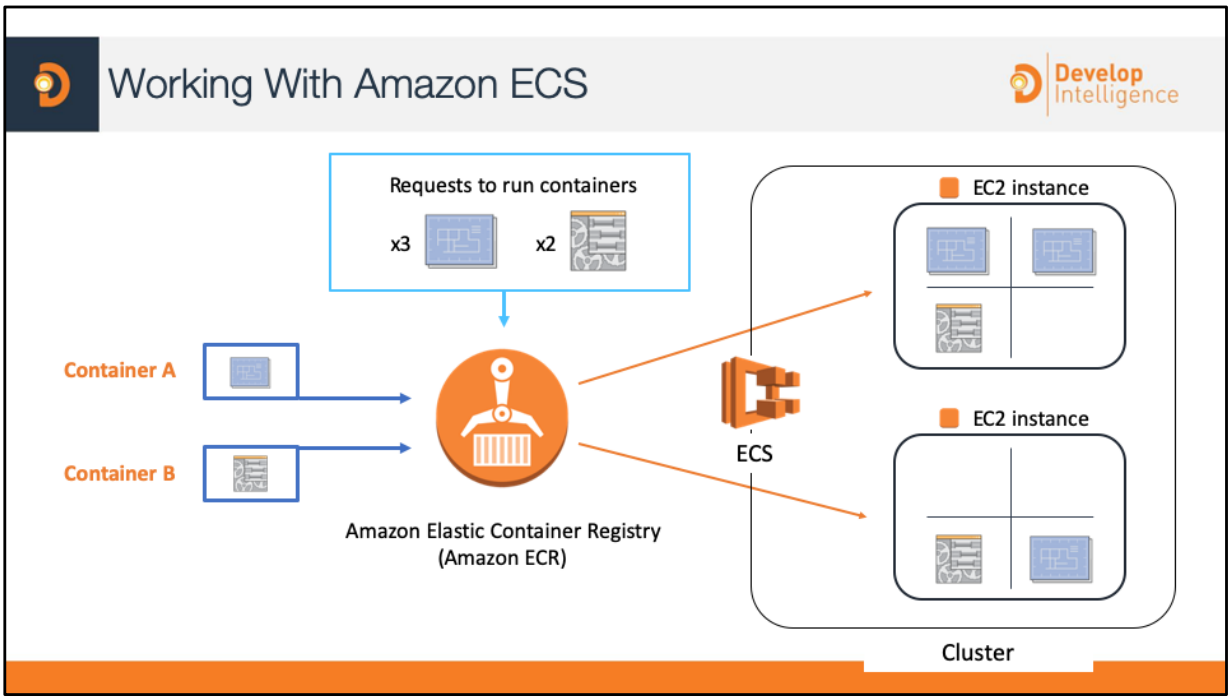


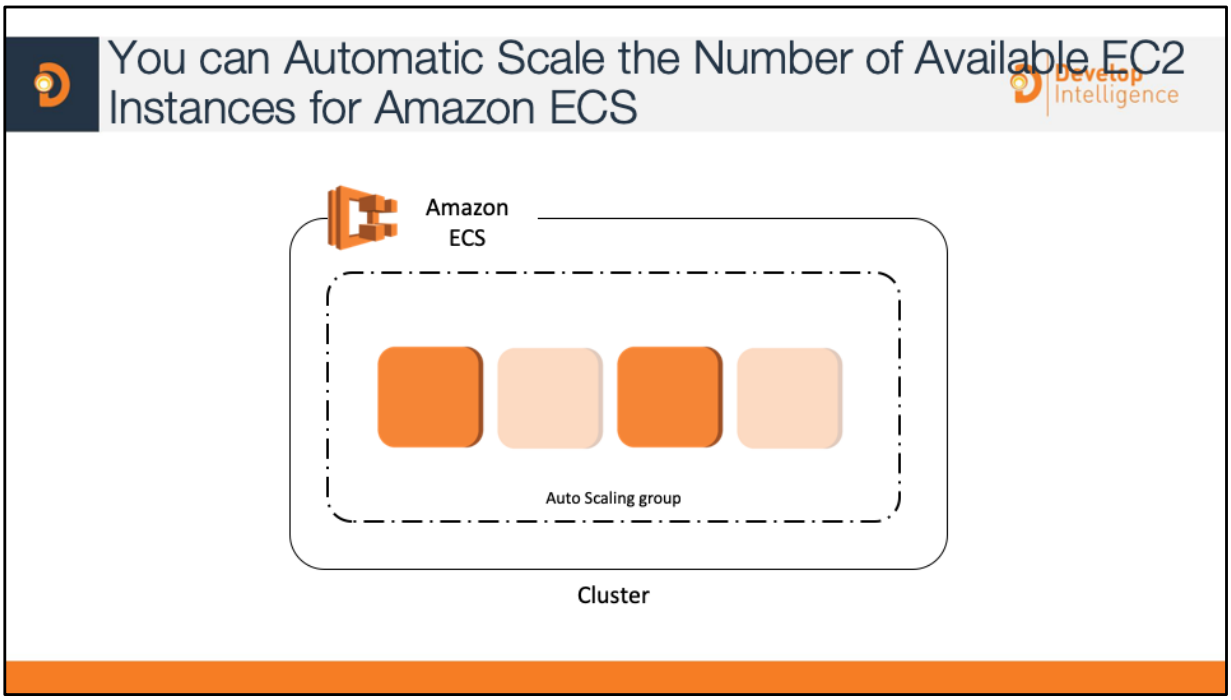
Amazon Elastic Container Service (Amazon ECS) is a highly scalable, high-performance container management service that supports Docker containers and allows you to easily run applications on a managed cluster of Amazon EC2 instances.

Amazon ECS is a scalable cluster service for hosting containers that:

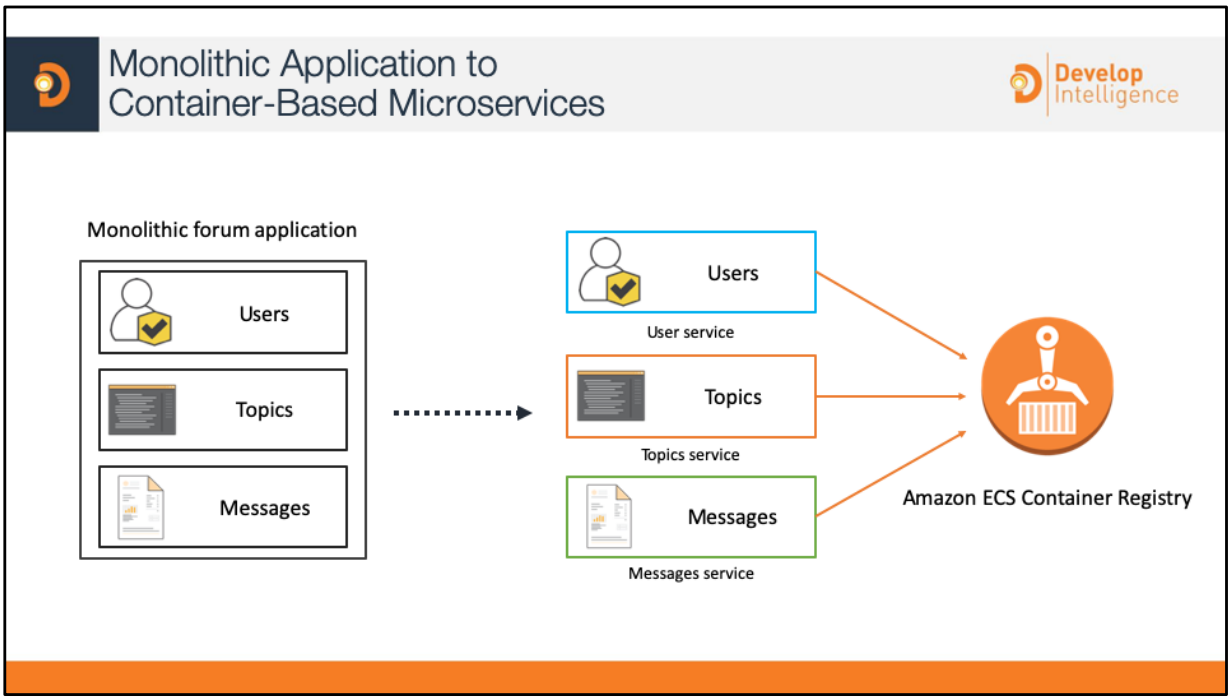
- Can scale up to thousands of instances
- Monitors deployment of containers
- Manages complete state of cluster
- Schedules containers using built-in scheduler or a third-party scheduler (e.g., Apache Mesos, Blox)
- Is extensible by using APIs

Clusters can leverage Spot and Reserved Instances.

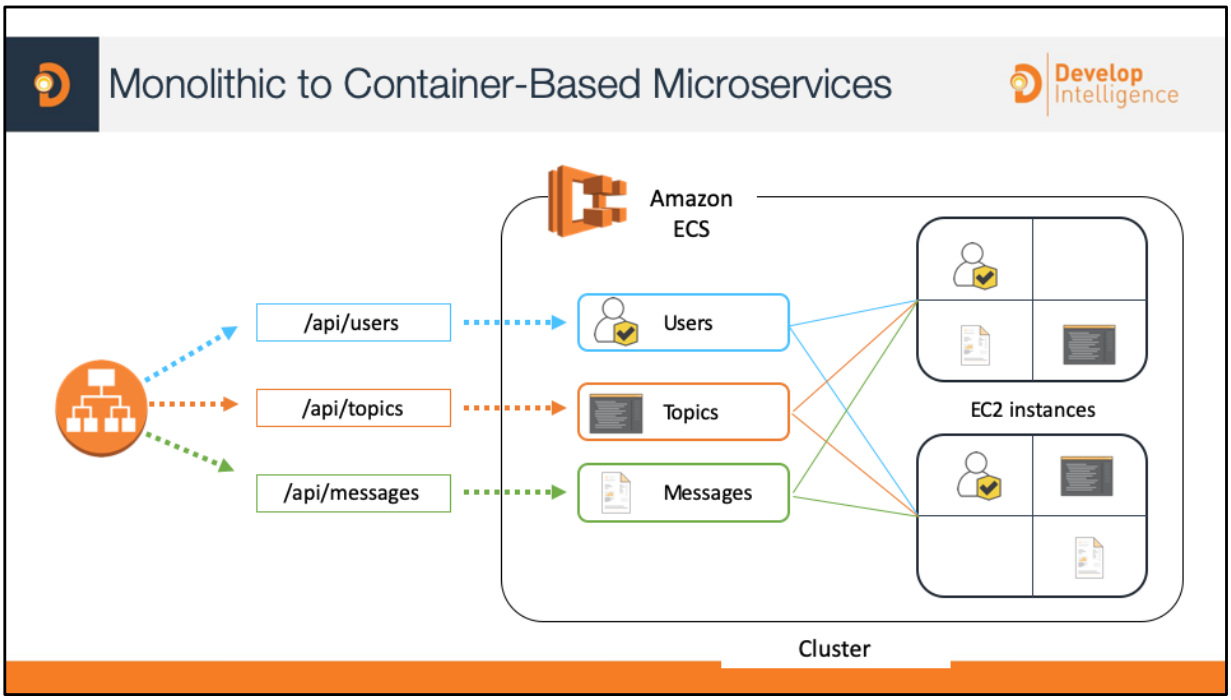

Working With Amazon ECS
Develop Intelligence
Requests to run containers
x3
x2
Container A
Container B
Amazon Elastic Container Registry
(Amazon ECR)
ECS
EC2 instance
EC2 instance
Cluster



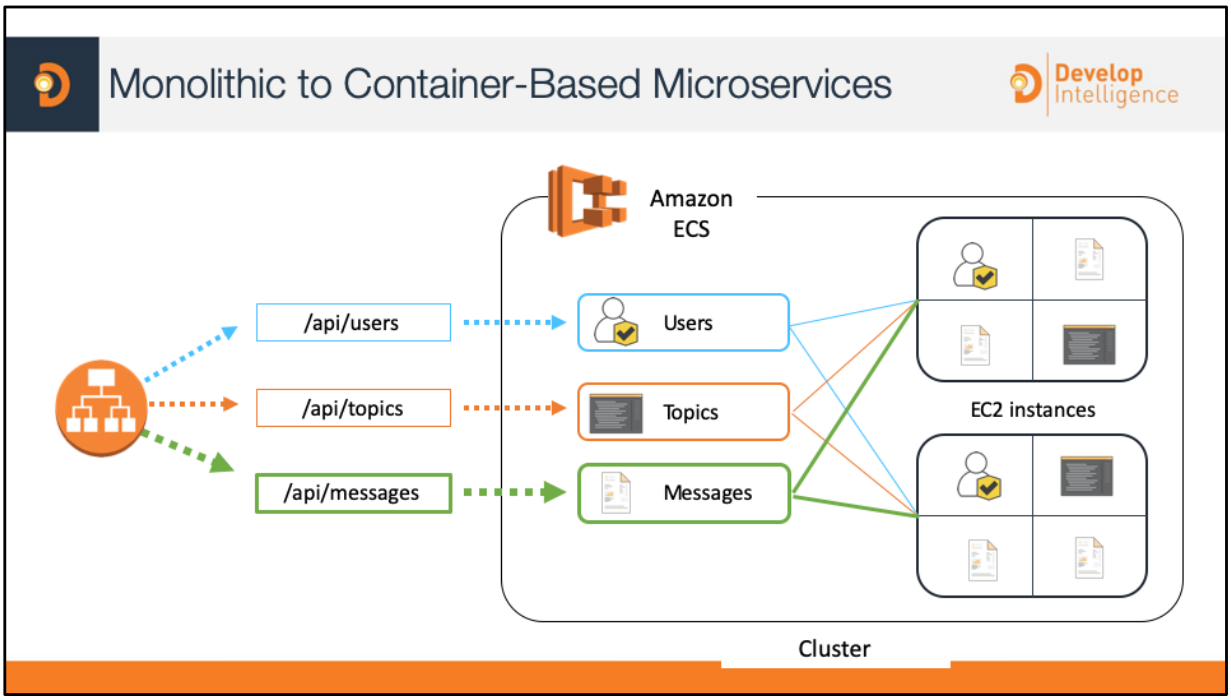
If you configure your Auto Scaling group to remove container instances, any tasks running on the removed container instances are killed. If your tasks are running as part of a service, Amazon ECS restarts those tasks on another instance if the required resources are available (CPU, memory, ports); however, tasks that were started manually will are not restarted automatically.



To shift this monolithic forum application into a microservice approach, you can break apart the code into individual encapsulated services. Make sure these each perform their function perfectly, and then register these services with the Amazon ECS Container Registry.



Next, inside Amazon ECS, create a service for each of these pieces of your original application. Then register the target group instances for these services. Finally, create an Application Load Balancer with target groups that point toward your Amazon ECS app services.



AWS Cloud Map and AWS App Mesh can assist you in building and troubleshooting your architectures.

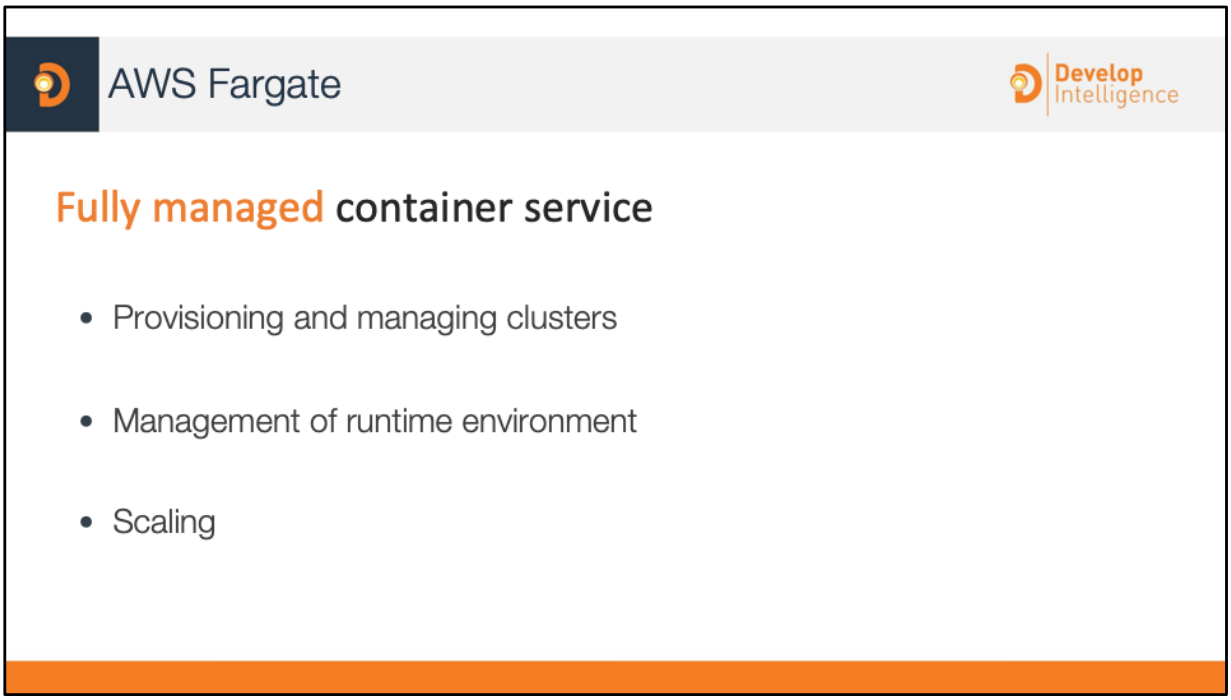
AWS Cloud Map is a fully managed service that allows you to register any application resources (such as databases, queues, microservices, and other cloud resources) with custom namespaces. AWS Cloud Map then constantly checks the health of resources to make sure the location is up-to-date, allowing you to add and register any resource with minimal manual intervention of mappings. AWS Cloud Map assists with service discovery, continuous integration, and health monitoring of your microservices and applications. For more information, see:

- https://aws.amazon.com/blogs/aws/aws-cloud-map-easily-create-and-maintain-custom-maps-of-your-applications/
- https://aws.amazon.com/cloud-map/
- https://www.youtube.com/watch?v=qTE1PbdY3hY

AWS App Mesh captures metrics, logs, and traces from every microservice which you can export to Amazon CloudWatch, AWS X-Ray, and compatible AWS partner and community tools for monitoring and tracing. It also provides custom control over traffic routing between microservices to assist with deployments, failures, or scaling of your application.

App Mesh lets you configure microservices to connect directly to each other via a proxy instead of requiring code within the application or using a load balancer. App Mesh uses Envoy, an open source service mesh proxy which is deployed alongside your microservice containers. For more information, see:

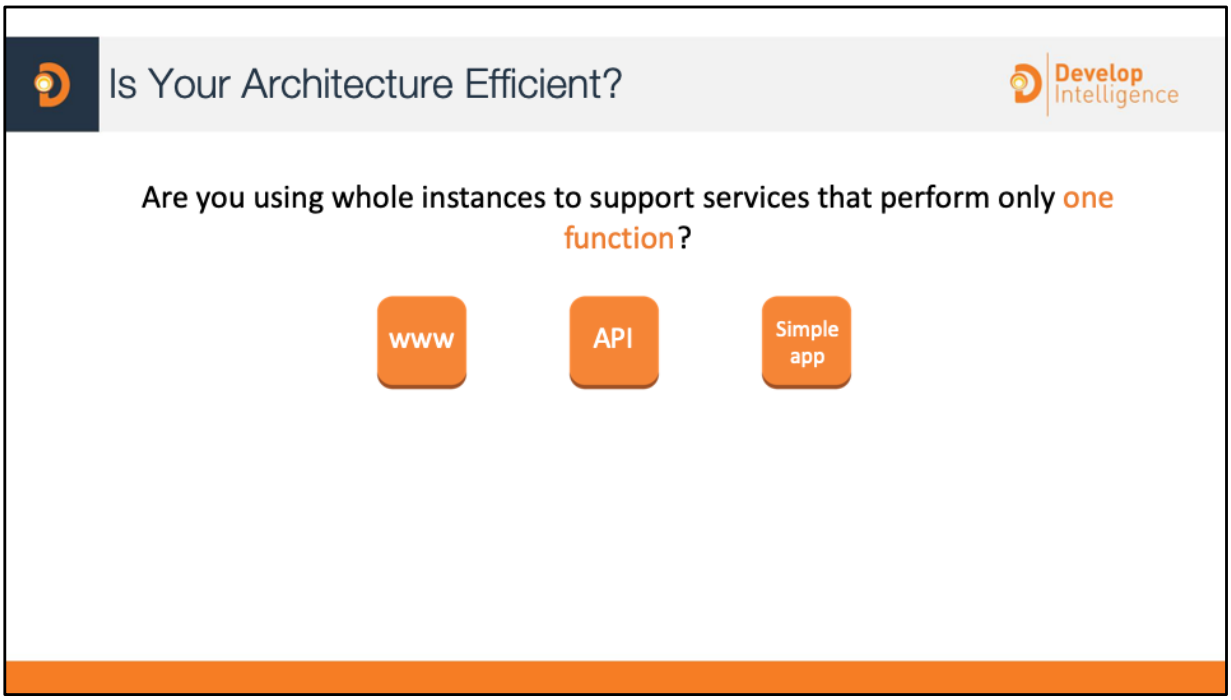
- https://aws.amazon.com/app-mesh/features/
- https://www.youtube.com/watch?v=qTE1PbdY3hY

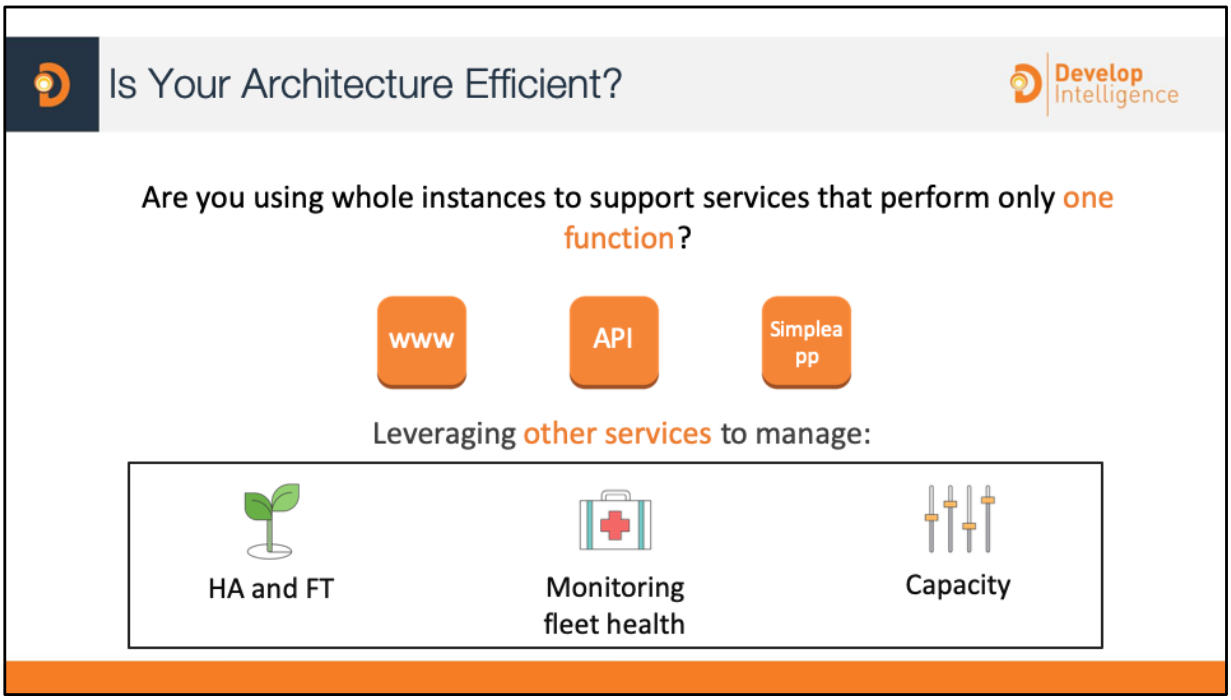


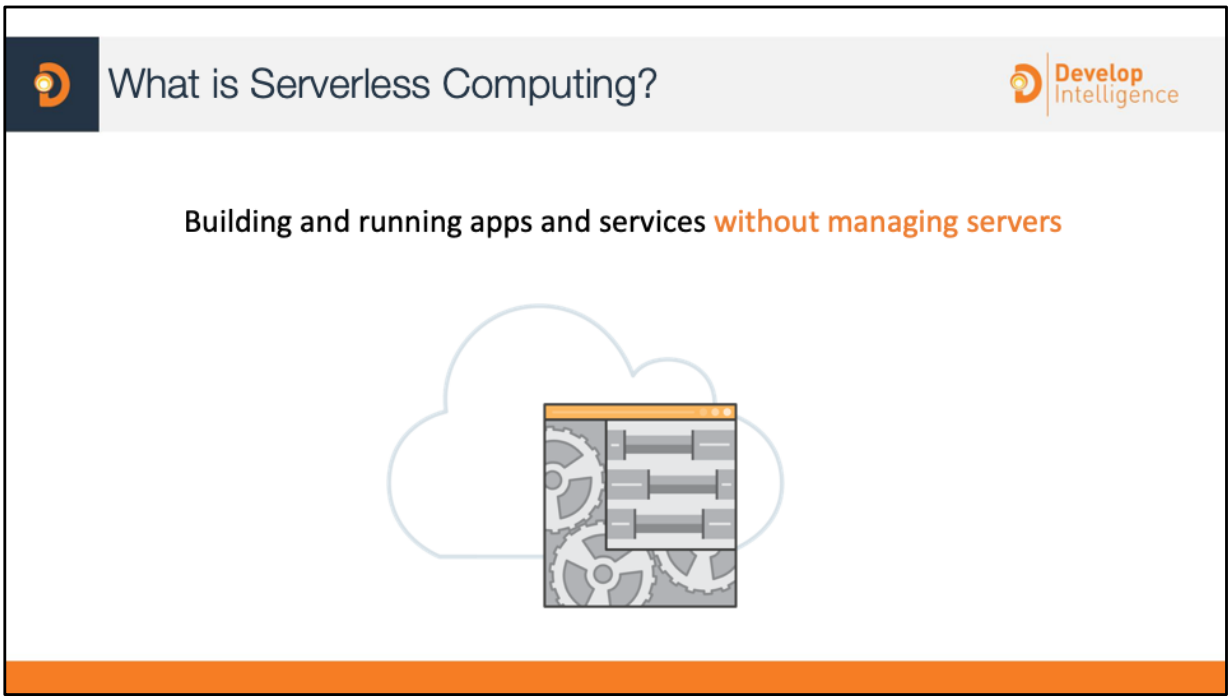
AWS Fargate is a technology for Amazon ECS and Amazon Elastic Container Service for Kubernetes (Amazon EKS) that allows you to run containers without having to manage servers or clusters. With AWS Fargate, you no longer have to provision, configure, and scale clusters of virtual machines to run containers. This removes the need to choose server types, decide when to scale your clusters, or optimize cluster packing. AWS Fargate eliminates the need for you to interact with or think about servers or clusters. Fargate lets you focus on designing and building your applications instead of managing the infrastructure that runs them.

The service supports both Amazon EKS and Amazon ECS.

For more information about AWS Fargate, see https://aws.amazon.com/fargate/

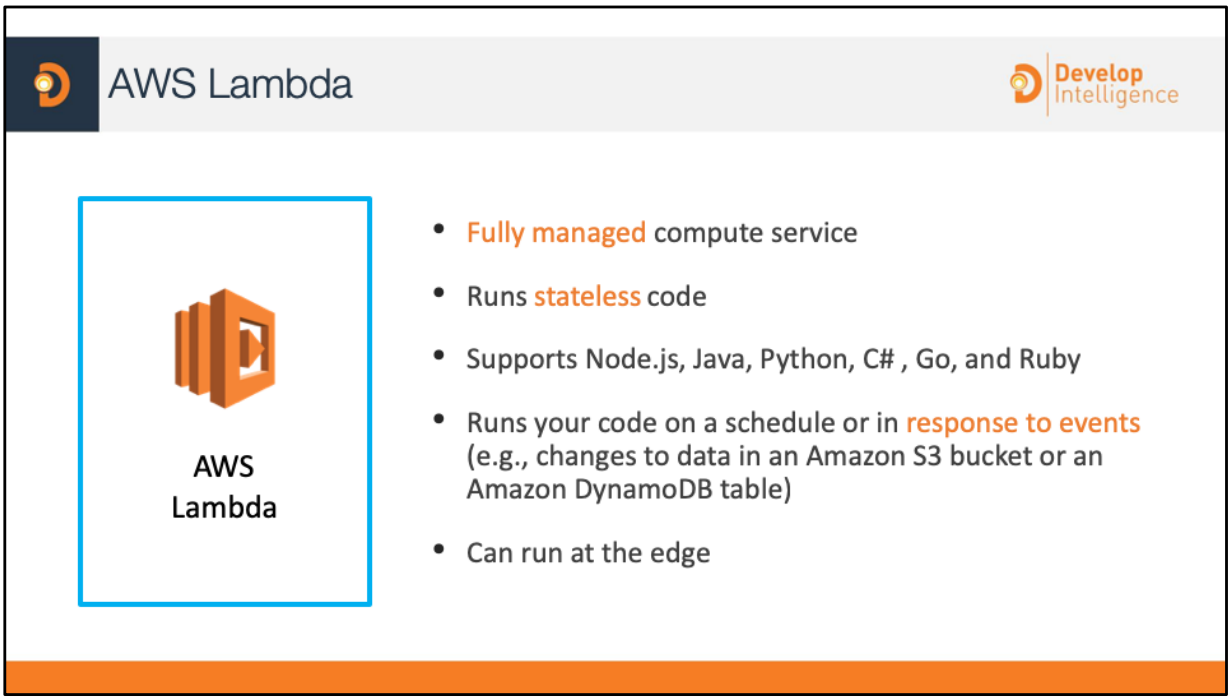

Is Your Architecture Efficient?
Develop Intelligence
Are you using whole instances to support services that perform only one function?
www
API
Simple app


Is Your Architecture Efficient?
Develop Intelligence
Are you using whole instances to support services that perform only one function?
www
API
Simplea pp
Leveraging other services to manage:
HA and FT
Monitoring fleet health
Capacity



What is serverless computing?
Serverless computing allows you to build and run applications and services without thinking about servers. Serverless applications don't require you to provision, scale, and manage any servers. You can build them for nearly any type of application or backend service, and everything required to run and scale your application with high availability is handled for you.

Why use serverless computing?
Building serverless applications means that your developers can focus on their core product instead of worrying about managing and operating servers or runtimes, either in the cloud or on-premises. This reduced overhead lets developers reclaim time and energy that can be spent on developing great products which scale and that are reliable.



AWS Lambda lets you run code without provisioning or managing servers. The service runs your code on a high-availability compute infrastructure and performs all of the administration of the compute resources, including server and operating system maintenance, capacity provisioning and automatic scaling, code monitoring and logging. All you need to do is supply your code in one of the languages that AWS Lambda supports (currently Node.js, Java, C#, Python, and Ruby).

Lambda@Edge provides the ability to execute a Lambda function in response to events generated by the Amazon CloudFront CDN and scale your code with high availability at an AWS edge location closest to end users. You can use Lambda functions to change CloudFront requests and responses at the following points:

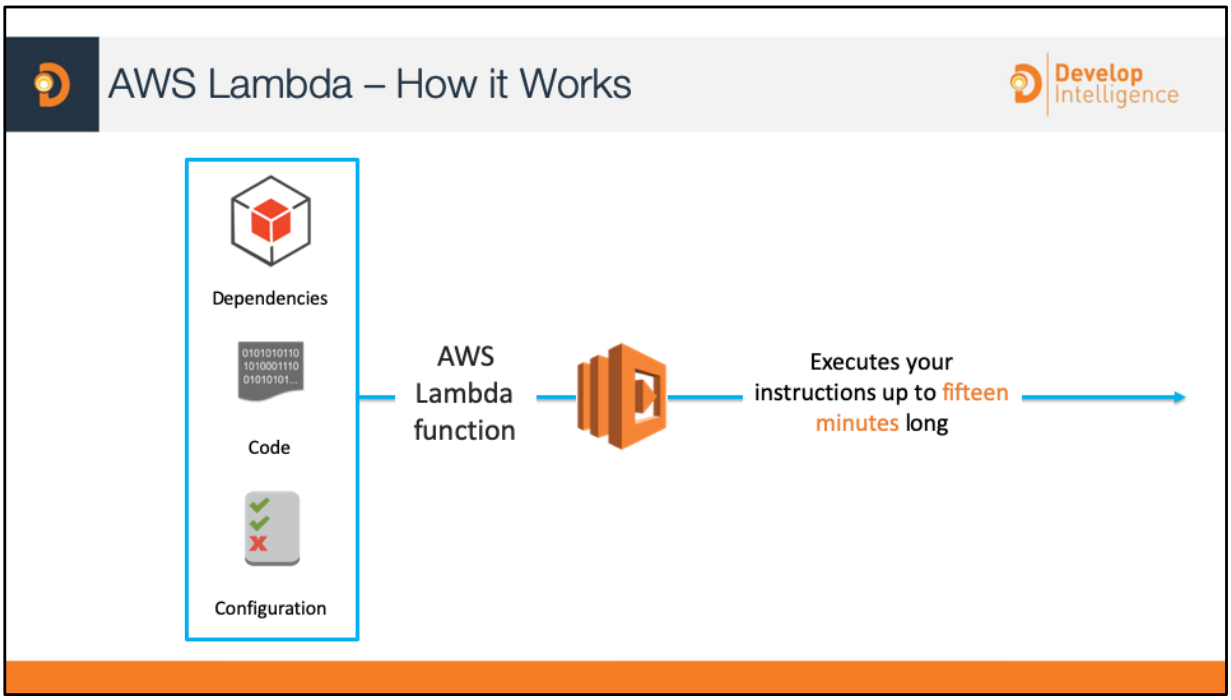
- Viewer request
- Origin request
- Origin response
- Viewer response

…to increase website security and privacy, build dynamic applications at the edge, SEO, real-time image transformation, user authentication and authorization, user tracking and analytics and other use cases.

Lambda@Edge only support Node.js

For more information, see:

- https://aws.amazon.com/lambda/edge/
  https://docs.aws.amazon.com/lambda/latest/dg/lambda-edge.html
- https://aws.amazon.com/blogs/networking-and-content-delivery/adding-http-security-headers-using-lambdaedge-and-amazon-cloudfront/



The core components of AWS Lambda are the *event source* and the *Lambda function*. Event sources publish events, and a Lambda function is the custom code that you write to process the events. Lambda executes your Lambda function on your behalf.

A Lambda function consists of your code, associated dependencies, and configuration. Configuration includes information such as the handler that will receive the event, the IAM role AWS Lambda can assume to execute the Lambda function on your behalf, the compute resource you want allocated, and the execution timeout.

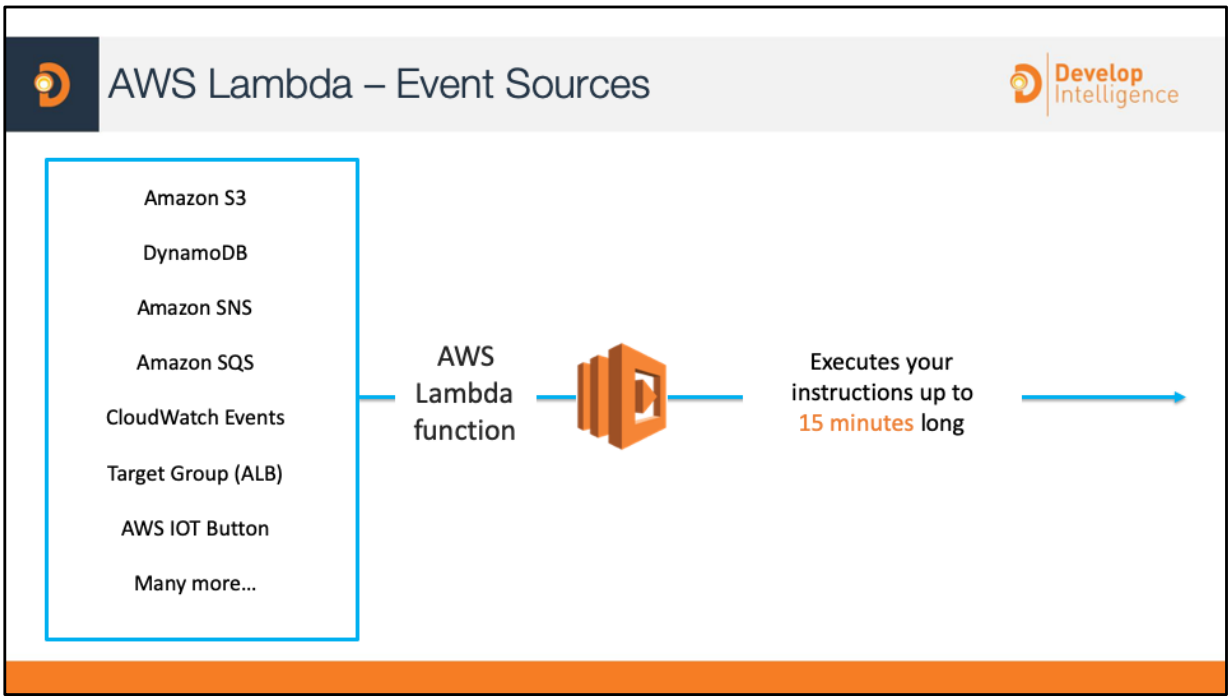
Layers enable AWS Lambda function developers to maintain packages, binaries, runtimes, and other files needed for their Lambda functions as components separate from their function code. When creating a Lambda function, you have the option to specify one or more layers that will be included with the function's execution environment. This removes the need to maintain copies of the same files that are distributed across multiple Lambda functions. For example, a serverless application written in Python could use a package such as PyMySQL to query an Amazon RDS MySQL database. With layers, only a single copy of the PyMySQL package needs to be maintained, and can be consumed by any function in the application.

Lambda Layer Restrictions:

- A single function can consume up to 5 layers at a time.
- The total unzipped size of the function (including layers) can't exceed the unzipped deployment package size limit of 250 MB.

Lambda layers support resource-level permissions, and can be shared across specific AWS accounts, AWS Organizations, or all accounts. Layers can be added to a function either during creation or later, and can be updated as needed. The AWS Serverless Application Model (SAM) also supports management of layers across functions.

Similar to function versions, layers support individual versions and corresponding permissions. A published version of a layer cannot be updated (other than changing the version permissions). To update a layer, a new version must be published. This way, you can control the rollout of new layers across multiple functions.



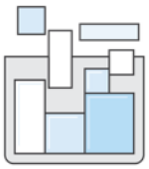
You can use the ALB to send traffic to your Lambda functions via HTTP/HTTPS. Or, because it is content-based routing, you can choose to send traffic to different Lambda functions based on a host or a host and URL path in the requests that come to the ALB. By registering your Lambda functions as targets to your ALB, the load balancer forwards the content to your Lambda function in a JSON format. By default, health checks are disabled for target groups of type lambda.
https://docs.aws.amazon.com/elasticloadbalancing/latest/application/lambda-functions.html

Benefits of Serverless Computing

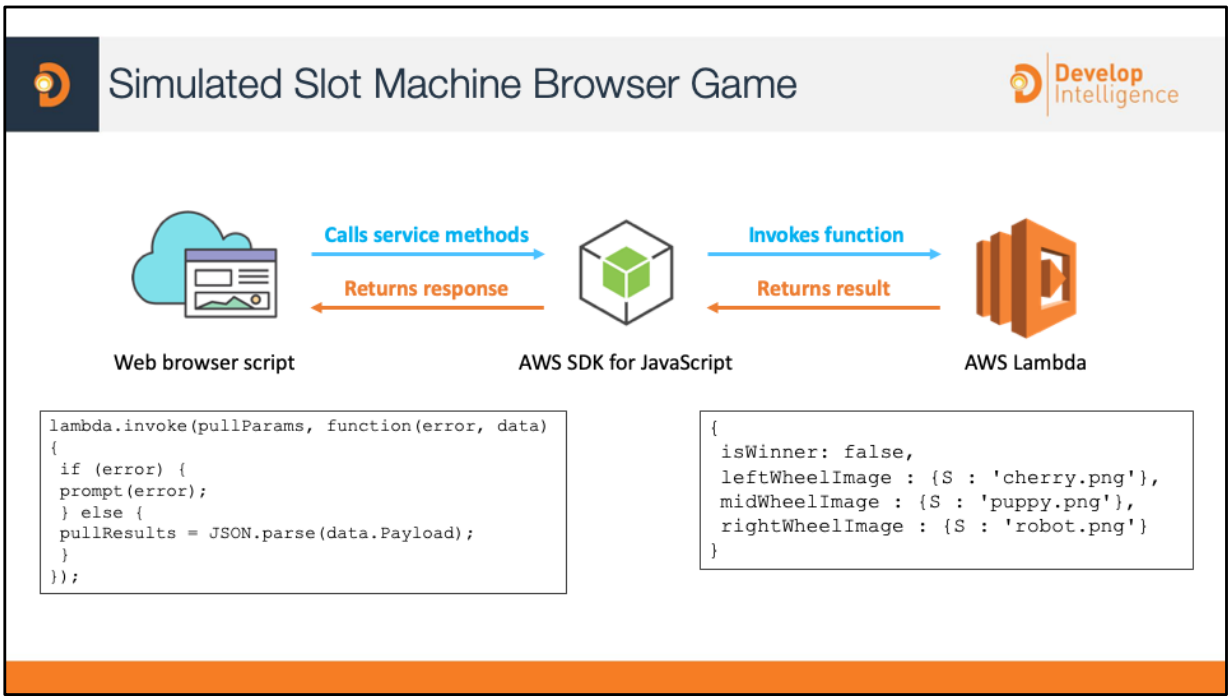
Develop Intelligence

Focus on your application, not configuration

Use compute resources only upon request

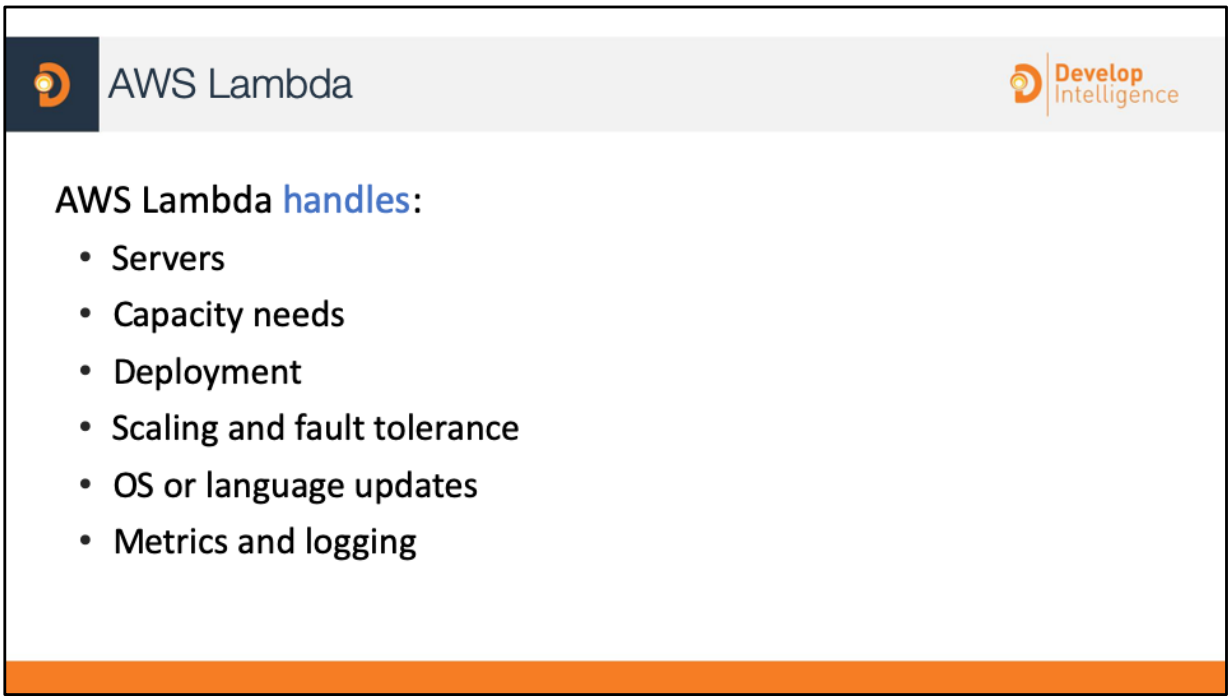


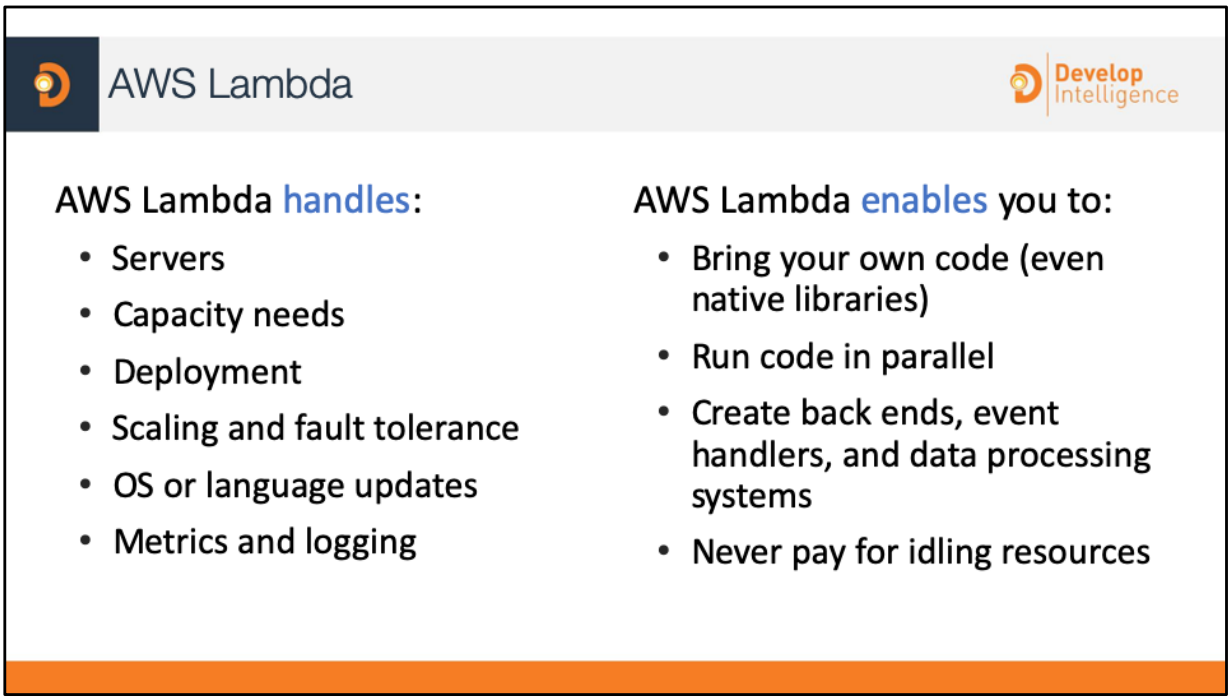
Build a microservice architecture



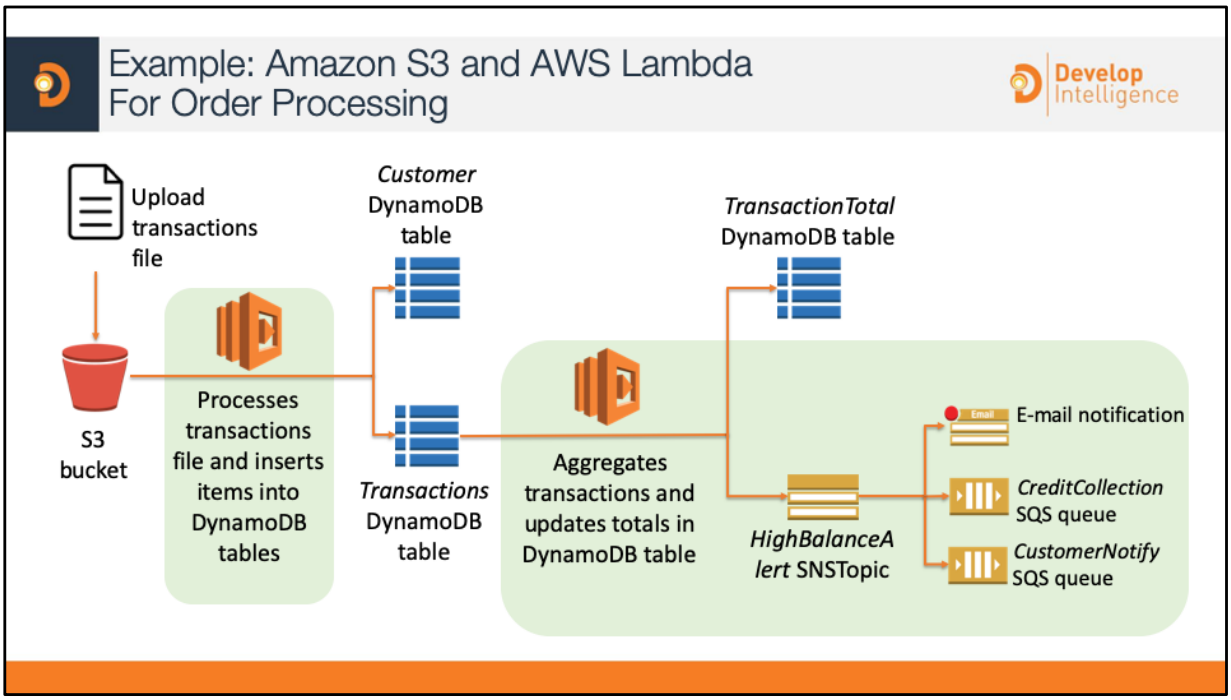
In this example, a simulated slot machine browser-based game invokes a Lambda function that generates the random results of each slot pull, returning those results as the file names of images used to display the result. The images are stored in an Amazon S3 bucket that is configured to function as a static web host for the HTML, CSS, and other assets needed to present the application experience.

For more information, see https://docs.aws.amazon.com/sdk-for-javascript/v2/developer-guide/lambda-examples.html


AWS Lambda
Develop Intelligence
AWS Lambda handles:
• Servers
• Capacity needs
• Deployment
• Scaling and fault tolerance
• OS or language updates
• Metrics and logging


AWS Lambda
Develop Intelligence
AWS Lambda handles:
• Servers
• Capacity needs
• Deployment
• Scaling and fault tolerance
• OS or language updates
• Metrics and logging
AWS Lambda enables you to:
• Bring your own code (even native libraries)
• Run code in parallel
• Create back ends, event handlers, and data processing systems
• Never pay for idling resources

Example: Amazon S3 and AWS Lambda For Order Processing



Amazon API Gateway

Develop Intelligence

API Gateway

Allows you to create APIs that act as "front doors" for your applications

Handles up to hundreds of thousands of concurrent API calls

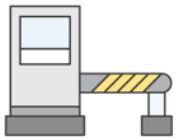
Can handle workloads running on:

- Amazon EC2
- AWS Lambda
- Any web application

API Gateway Protects You

Develop Intelligence

API Gateway



Prevents exposing endpoints

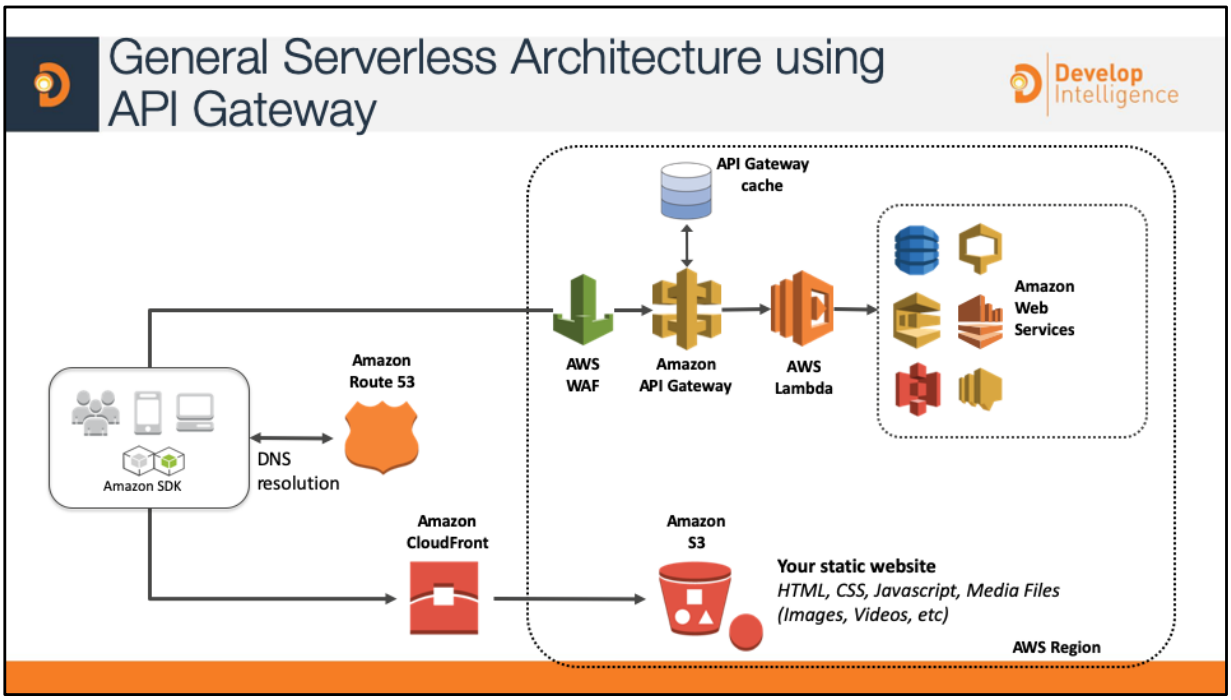
Protection from DDoS and injection attacks

API Gateway

Develop Intelligence

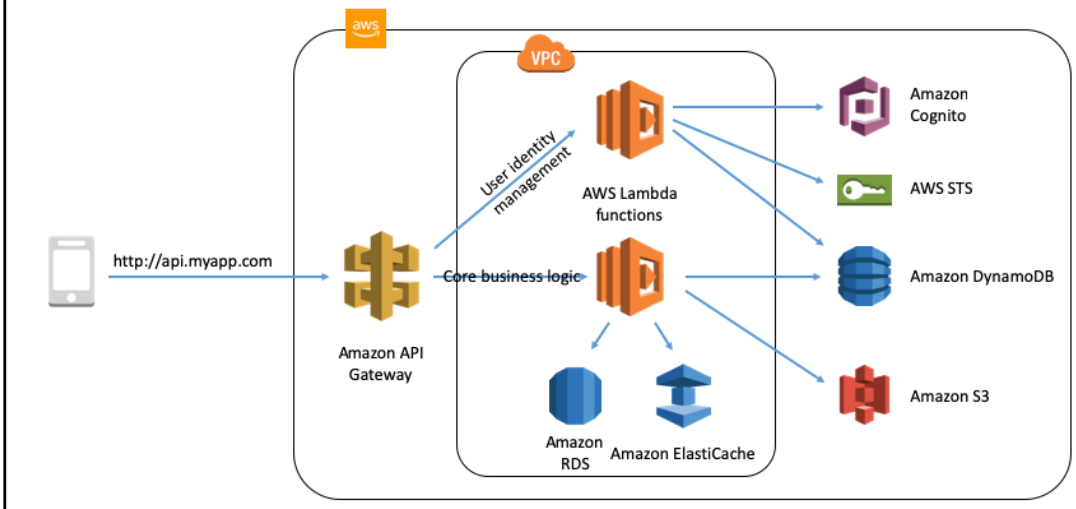
API Gateway

- Host and use multiple versions and stages of your APIs
- Create and distribute API keys to developers
- Leverage signature version 4 to authorize access to APIs
- Deeply integrated with AWS Lambda
- Endpoint integration with private VPCs



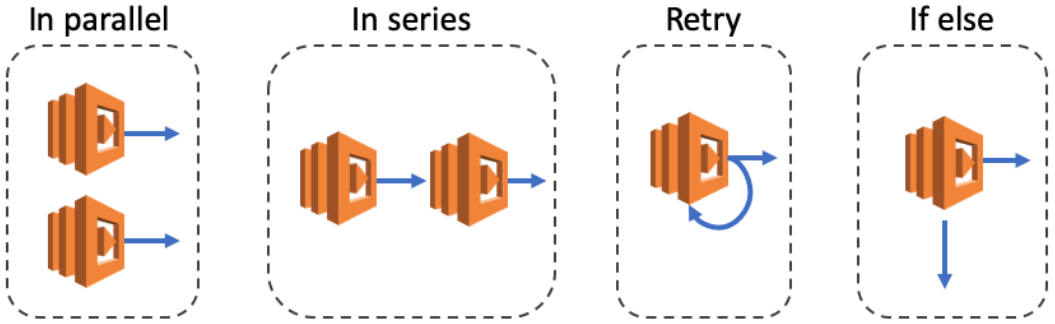
Containers introduce a new paradigm of software delivery, and Amazon ECS/Fargate make it easy to manage them—but you still need to stand up infrastructure and consume resources.

Serverless Mobile Backend



What If I Need Lambda or API Gateway To Act…

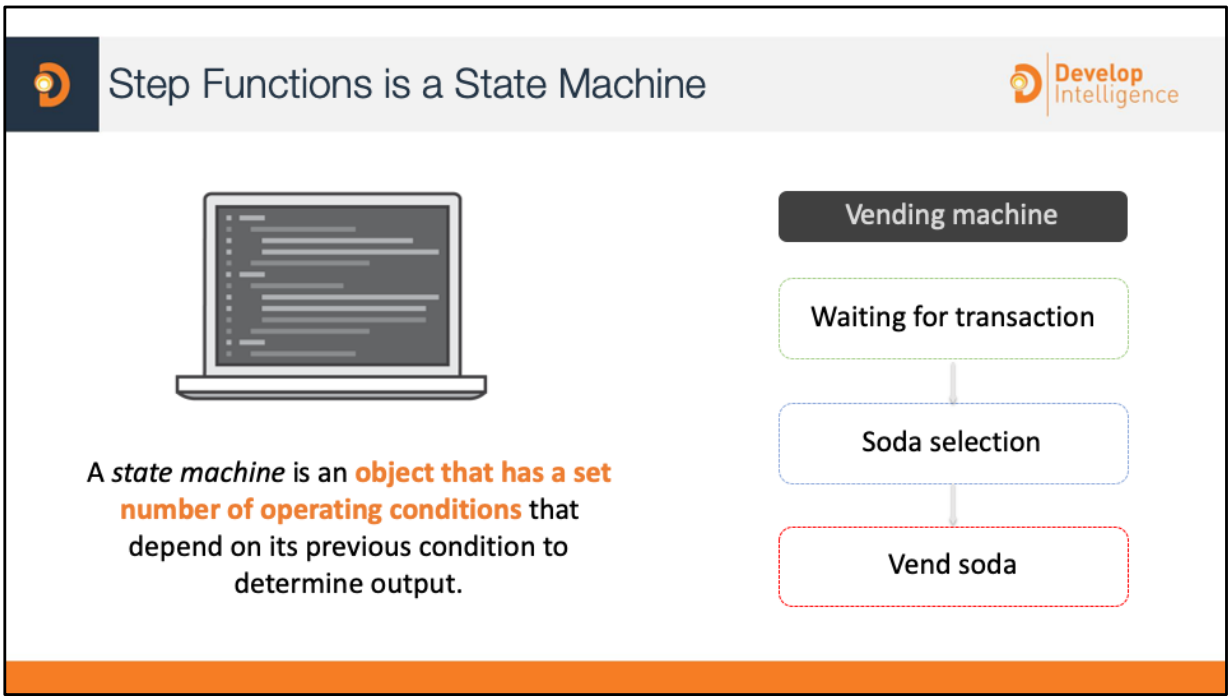
Develop Intelligence



AWS Step Functions

AWS Step Functions

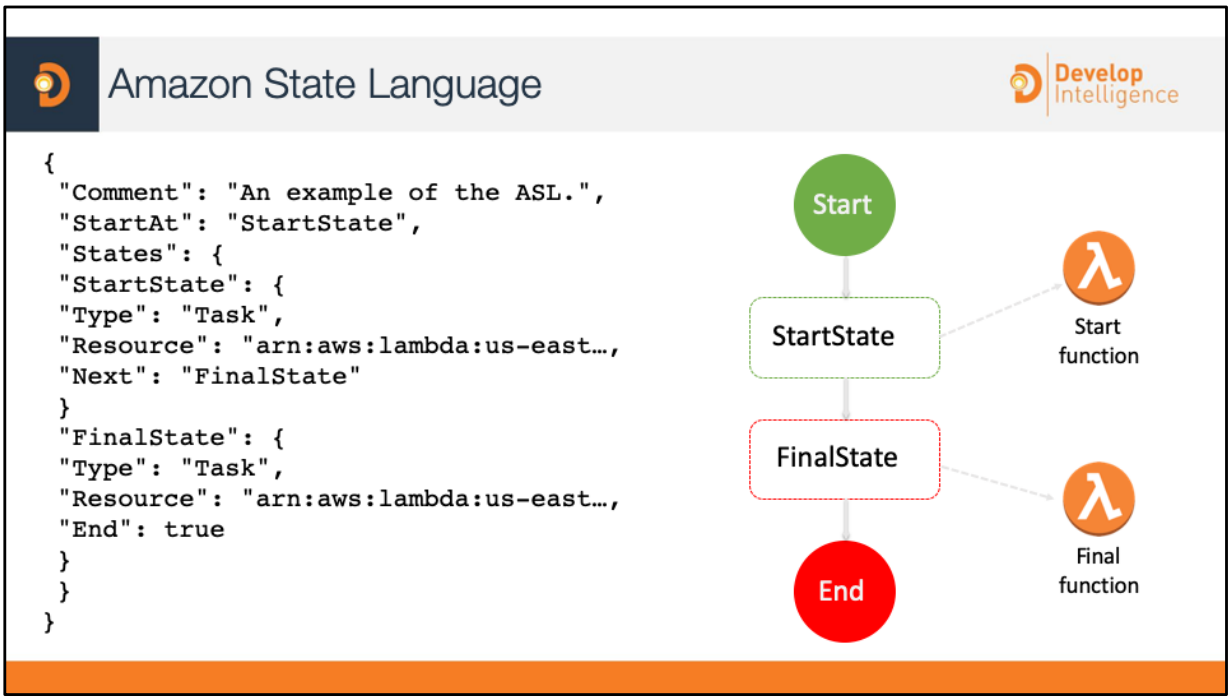
- Coordinates microservices using visual workflows
- Allows you to step through the functions of your application
- Automatically triggers and tracks each step
- Provides simple error catching and logging if a step fails



A *state machine* is an object that has a set number of operating conditions that depend on its previous condition to determine output.

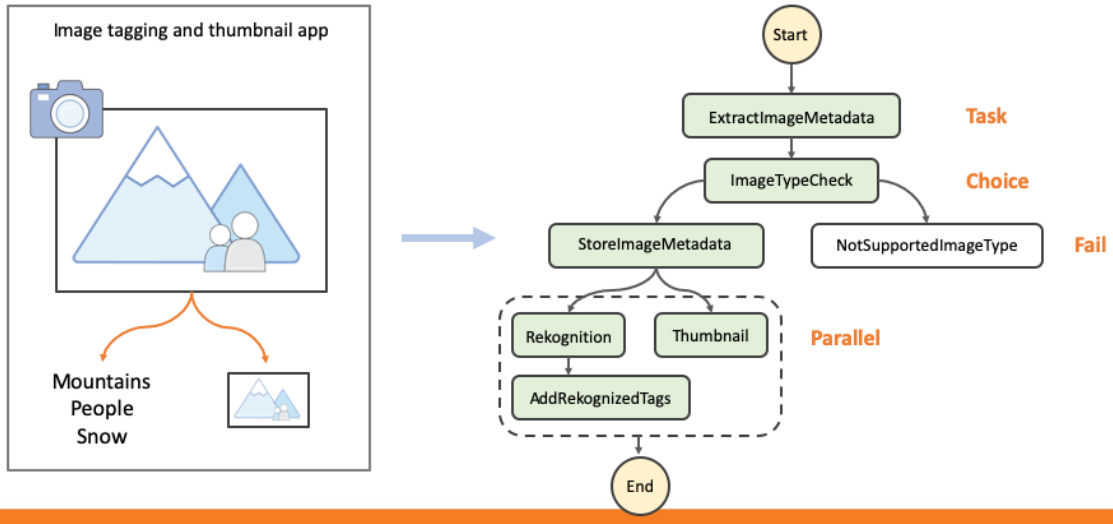
A common example of a state machine is the soda vending machine. The machine starts in the operating state (waiting for a transaction), and then moves to soda selection when money is added. After that, it enters a vending state, where soda is deployed to the customer. After completion, the state returns back to operating.

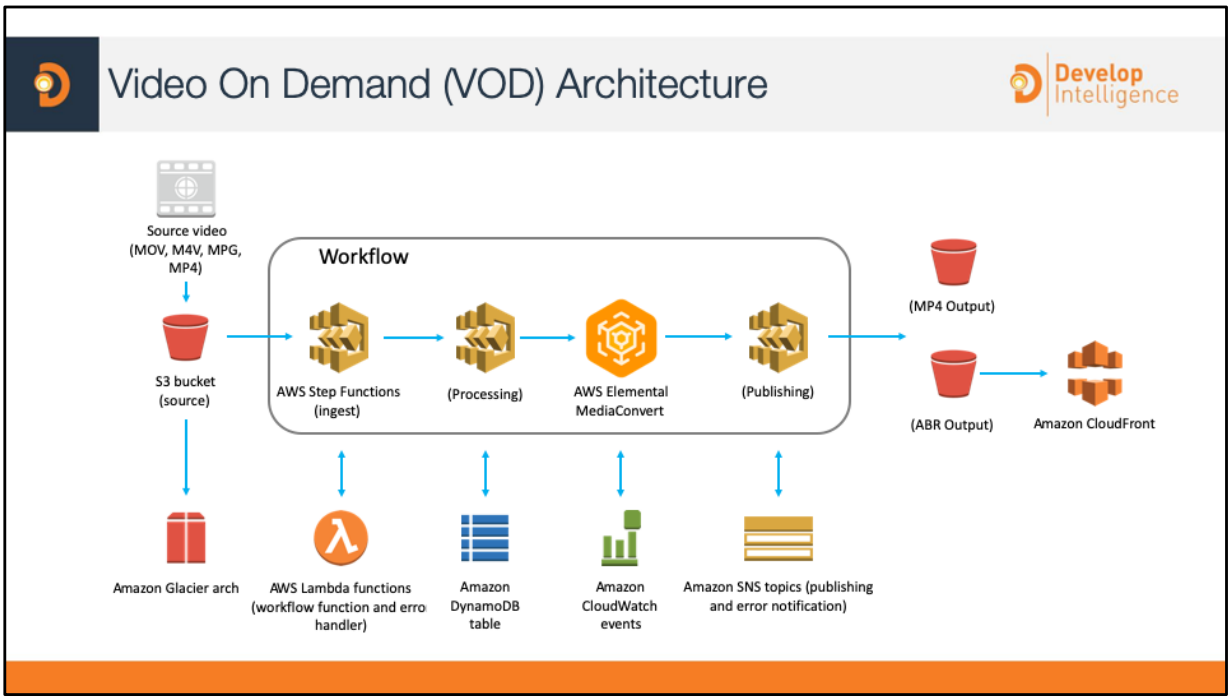
AWS Step Functions allows you to create and automate your own state machines within the AWS environment. It does this with the use of a JSON-based Amazon State Language, which contains a structure made of various states, tasks, choices, error handling and more.



Amazon States Language is a JSON-based, structured language used to define your state machine, a collection of states, that can do work (task states), determine which states to transition to next (Choicestates), stop an execution with an error (fail states), and so on. For more information, see the Amazon States Language Specification and Statelint, a tool that validates Amazon States Language code.

Build Visual Workflows Using State Types





AWS offers a solution that ingests source videos, processes the videos for playback on a wide range of devices, and stores the transcoded media files for on-demand delivery to end users through Amazon CloudFront. For more information, see https://docs.aws.amazon.com/solutions/latest/video-on-demand/architecture.html

If you would like to use Amazon Elastic Transcoder for encoding, this video-on-demand solution includes another AWS CloudFormation template that deploys the same workflow with Elastic Transcoder. For more information about that, see https://docs.aws.amazon.com/solutions/latest/video-on-demand/appendix-e.html

Lab 5:

Implementing a Serverless Architecture

Develop Intelligence