What Is Serverless

- Traditional On-Premise
  - You predict workloads, budget and purchase hardware to meet maximum demand, then build, test, deploy, and maintain.
- Then comes hosted servers
  - New servers can be set up on shorter notice, but you still have to deploy your services, test, monitor, and maintain them.
- Hosted servers go virtual
  - Deploying new servers now take minutes and can be instantiated from pre-built images. Servers can be scaled up and down responding to load, but must still be monitored and maintained.
- And then comes serverless
  - No management for either hardware or the operating system, and often not other things as well.

AWS Serverless Services

By Category

Compute	Docker, Lambda, Fargate
Storage	S3, EFS
Data Stores	DynamoDB, Aurora
Front-Ends	API Gateway, Elastic Load Balancer, Route53
Application Integration	SNS, SQS, AppSync
Orchestration	Step Functions
Analytics	Kinesis, Athena
CI/CD	CodeStar, CodePipeline, CodeBuild, CodeDeploy
Authoring	Cloud 9
Monitoring/Logging	CloudWatch, X-Ray

The Old Way: Docker

- A stripped-down virtual machine
- Build environment becomes the deployment environment
- Updated as a whole
- Based on layers

- Elastic Container Registry (ECR)
- Elastic Container Service (ECS)
- Elastic Kubernetes Service (EKS)
- Fargate

The New Way: Lambda

- Docker-type container stripped to its bare minimum
- A single disposable instance per function call
  - Infinitely scalable
  - Insulates runtime environment from bugs, bad data, attacks

Lambda Pricing

Memory (MB)	Free tier seconds per month	Price per 100ms ($)
128	3,200,000	0.000000208
192	2,133,333	0.000000313
256	1,600,000	0.000000417
320	1,280,000	0.000000521
384	1,066,667	0.000000625
448	914,286	0.000000729
512	800,000	0.000000834
576	711,111	0.000000938
640	640,000	0.000001042
704	581,818	0.000001146
...		
3008	136,170	0.000004897

Things What Been Done For You

- Data Stores: Aurora, DynamoDB, MySQL, PostGRESQL
- Storage: EFS, S3, FSx
- DNS: Route53
- Load Balancing: Application Load Balancer
- Front End: API Gateway
- Monitoring and Security: CloudWatch, CloudTrail, GuardRails, Inspector

Orchestration

- Simple Notification Service (SNS)
  - A managed pub/sub messaging service. Can push to SQS, Lambda, or end users (email, SMS, mobile push)
- Simple Queue Service (SQS)
  - A managed scalable messaging pipeline for communication between services
- AppSync
  - A managed service that uses GraphQL to enable applications to get the data they need

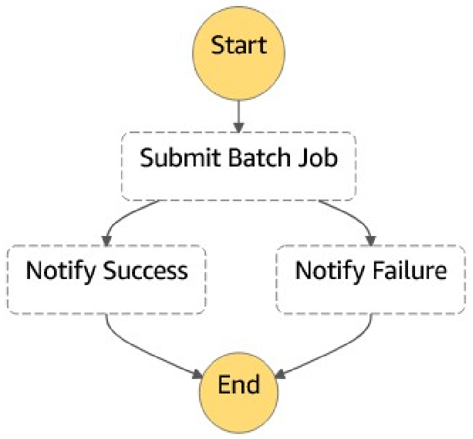
Building, Test, and Deploy

- Cloud 9
  - Managed IDE
- X-Ray
  - Trace and debug serverless applications
- CodeBuild
  - Managed and scalable build, test, package system
- CodeDeploy
  - Fully automates code updates to EC2, Lambda, and on-premise systems
- CodePipeline
  - Continuous integration and delivery service

Making It All Easier

- Step Functions
  - Coordinate the components of distributed applications and microservices using visual workflows
  - Building applications from individual components that each perform a discrete function lets you scale and change applications quickly
- CodeStar
  - Enables you to quickly develop, build, and deploy applications on AWS
  - Set up your entire continuous delivery toolchain in minutes

Step Functions - Example



```
{
  "Comment": "An example of the Amazon States Language for notification on an AWS Batch job completion",
  "StartAt": "Submit Batch Job",
  "TimeoutSeconds": 3600,
  "States": {
    "Submit Batch Job": {
      "Type": "Task",
      "Resource": "arn:<PARTITION>:states:::batch:submitJob.sync",
      "Parameters": {
        "JobName": "BatchJobNotification",
        "JobQueue": "<BATCH_QUEUE_ARN>",
        "JobDefinition": "<BATCH_JOB_DEFINITION_ARN>"
      },
      "Next": "Notify Success",
      "Catch": [
        {
          "ErrorEquals": [ "States.ALL" ],
          "Next": "Notify Failure"
        }
      ]
    },
    "Notify Success": {
      "Type": "Task",
      "Resource": "arn:<PARTITION>:states:::sns:publish",
      "Parameters": {
        "Message": "Batch job submitted through Step Functions succeeded",
        "TopicArn": "<SNS_TOPIC_ARN>"
      },
      "End": true
    },
    "Notify Failure": {
      "Type": "Task",
      "Resource": "arn:<PARTITION>:states:::sns:publish",
      "Parameters": {
        "Message": "Batch job submitted through Step Functions failed",
        "TopicArn": "<SNS_TOPIC_ARN>"
      },
      "End": true
    }
  }
}
```