DevOps and AWS

Presented by
VAISHALI TAPASWI

FANDS INFONET Pvt.Ltd.

www.fandsindia.com
fands@vsnl.com

Ground Rules

- **Turn off cell phone. If you cannot, please keep it on silent mode. You can go out and attend your call.**
- **If you have questions or issues, please let me know immediately.**
- **Let us be punctual.**

Agenda

What is Cloud Computing?

- A model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g. networks, server, storage, applications and services) that can be rapidly provisioned and released with minimal management effort of service provider interaction.

Cloud Computing at Glance

- 5 Essential Characters
- 3 Service Models
- 4 Deployment Models

5 Essential Characters

- On demand self services
- Broad network access
- Resource pooling
- Rapid elasticity
- Measured service
- Multi Tenacity

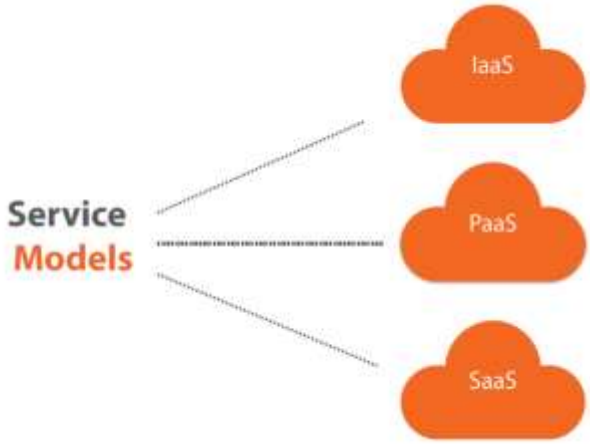
Essential Characters

- On-demand self-service
  - A consumer can unilaterally provision computing capabilities, such as server, Storage and network ,
  - As needed automatically without requiring human interaction with each service provider.
- Broad network access
  - Capabilities are available over the network and accessed through standard mechanisms that promote use by heterogeneous thin or thick client platforms
  - (e.g. mobile phones, tablets, laptops, and workstations).
- Resource pooling
  - The provider's computing resources are pooled to serve multiple consumers using a multi-tenant model, with different physical and virtual resources dynamically assigned and reassigned according to consumer demand.
  - There is a sense of location independence in that the customer generally has no control or knowledge over the exact location of the provided resources but may be able to specify location at a higher level of abstraction (e.g., country, state, or datacenter).
  - Examples of resources include storage, processing, memory, and network bandwidth.

Essential Characters

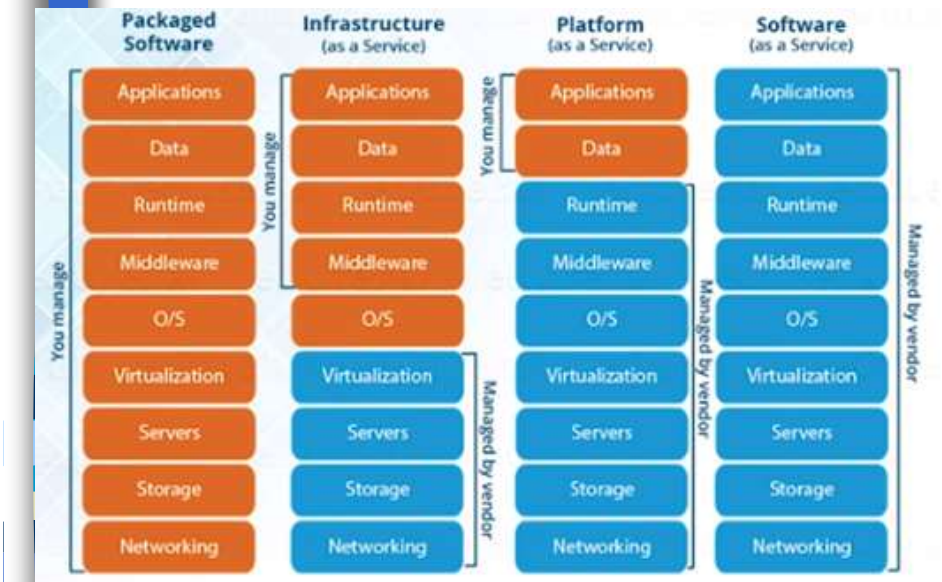
- Rapid elasticity
  - Capabilities can be elastically provisioned and released, in some cases automatically, to scale rapidly outward and inward commensurate with demand.
  - To the consumer, the capabilities available for provisioning often appear to be unlimited and can be appropriated in any quantity at any time.
- Measured service
  - Cloud systems automatically control and optimize resource use by leveraging a metering capability at some level of abstraction appropriate to the type of service (e.g. storage, processing, bandwidth, and active user accounts).
  - Resource usage can be monitored, controlled, and reported, providing transparency for both the provider and consumer of the utilized service.

3 Service Models



AS A SERVICE

- IAAS: INFRASTRUCTURE
  - Amazon Web Services (AWS), Cisco Metapod, Microsoft Azure, Google Compute Engine (GCE)
- PAAS: PLATFORM
  - Apprenda
- SAAS: SOFTWARE
  - Google Apps, Concur, Citrix GoToMeeting, Cisco WebEx



fands™
A Fast AND Study Approach

AWS and DevOps

AWS and DevOps

- AWS provides a set of flexible services designed to enable companies to more rapidly and reliably build and deliver products using AWS and DevOps practices. These services simplify provisioning and managing infrastructure, deploying application code, automating software release processes, and monitoring your application and infrastructure performance.

Main Features

- Get Started Fast
- Fully Managed Services
- Built for Scale
- Programmable
- Automation
- Secure

DevOps Tooling by AWS

- AWS provides services that help you practice DevOps at your company and that are built first for use with AWS. These tools automate manual tasks, help teams manage complex environments at scale, and keep engineers in control of the high velocity that is enabled by DevOps.

DevOps <->AWS

DevOps <->AWS

www.fandsindia.com

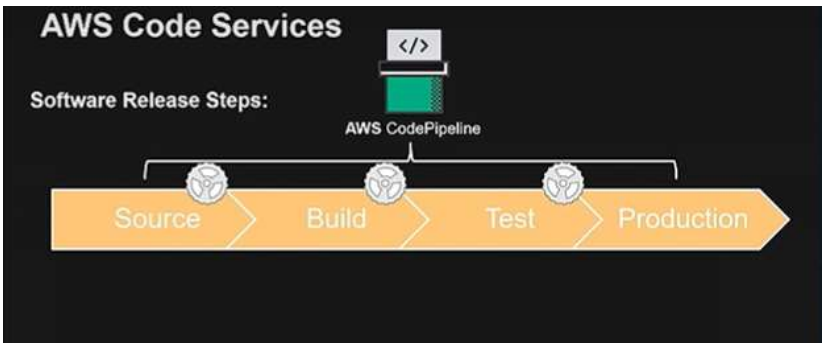
DevOps <->AWS

www.fandsindia.com

DevOps <->AWS

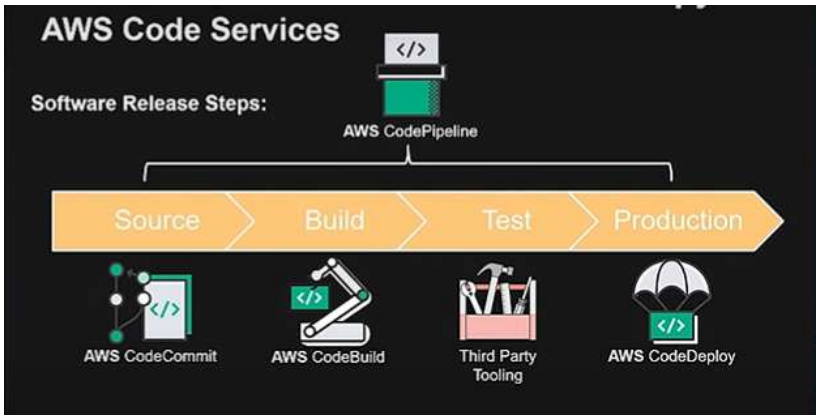
www.fandsindia.com

DevOps <->AWS



www.fandsindia.com

DevOps <->AWS



Introducing: AWS CodeStar

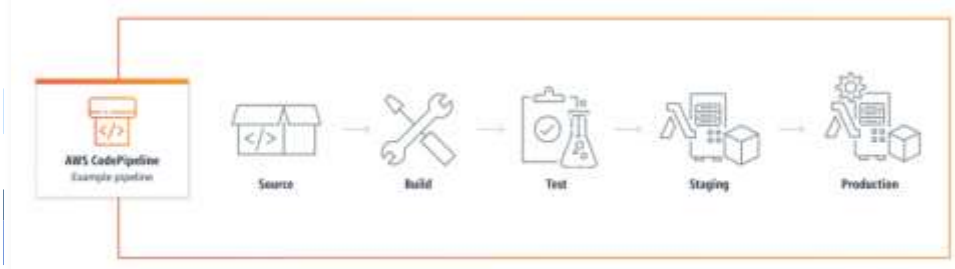
- Quickly develop, build and deploy applications on AWS
- Work across your team, securely
- Manage software delivery easily
- Choose from a variety of project templates

Introducing CodeStar

- Quickly develop, build and deploy applications on AWS
- Work across your team, securely
- Manage software delivery easily
- Choose from a variety of project templates

AWS CodePipeline

- AWS CodePipeline is a fully managed continuous delivery service that helps you automate your release pipelines for fast and reliable application and infrastructure updates.



AWS Data Pipeline

AWS Data Pipeline

- AWS Data Pipeline is a web service that you can use to automate the movement and transformation of data. With AWS Data Pipeline, you can define data-driven workflows, so that tasks can be dependent on the successful completion of previous tasks. You define the parameters of your data transformations and AWS Data Pipeline enforces the logic that you've set up.

Components

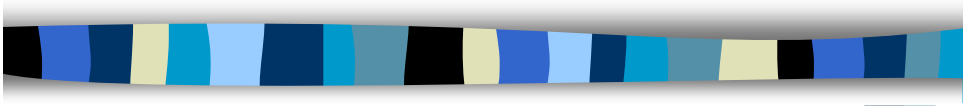
- Components Pipeline
  - A pipeline definition specifies the business logic of your data management. For more information, see Pipeline definition file syntax.
  - A pipeline schedules and runs tasks by creating Amazon EC2 instances to perform the defined work activities.
  - Task Runner polls for tasks and then performs those tasks. For example, Task Runner could copy log files to Amazon S3 and launch Amazon EMR clusters.

Pipeline Objects

- ShellCommandActivity
  - Reads the input log file and counts the number of errors.
- S3DataNode (input)
  - The S3 bucket that contains the input log file.
- S3DataNode (output)
  - The S3 bucket for the output.
- Ec2Resource
  - The compute resource that AWS Data Pipeline uses to perform the activity.
    - Note that if you have a large amount of log file data, you can configure your pipeline to use an EMR cluster to process the files instead of an EC2 instance.
- Schedule
  - Defines that the activity is performed every 15 minutes for an hour.

fands™
A Fast AND Steady Approach

Lambda



fands™

Serverless on AWS

- Build & Run apps without thinking about servers
- Serverless is a way to describe the services, practices, and strategies that enable you to build more agile applications so you can innovate and respond to change faster.
- Infrastructure management tasks like capacity provisioning and patching are handled by AWS.
- Serverless services like AWS Lambda come with automatic scaling, built-in high availability, and a pay-for-value billing model.

Benefits

- Move from idea to market, faster
  - By eliminating operational overhead, your teams can release quickly, get feedback, and iterate to get to market faster.
- Lower your costs
  - pay-for-value billing, no need to over-provision
- Adapt at scale
  - Automatically scale from zero to peak demands,
- Build better applications, easier
  - Serverless applications have built-in service integrations, so you can focus on building your application instead of configuring it.

Serverless Services on AWS

- Compute
  - AWS Lambda
  - Amazon Fargate
- Data Store
  - Amazon S3
  - Amazon DynamoDB
  - Amazon RDS Proxy
  - Amazon Aurora Serverless
- Application Integration
  - Amazon EventBridge
  - AWS Step Functions
  - Amazon SQS
  - Amazon SNS
  - Amazon API Gateway
  - AWS AppSync

Lambda

- Run code without provisioning or managing servers and pay only for the resources you consume
- Benefits
  - No servers to manage
  - Continuous scaling
  - Cost optimized with millisecond metering
  - Consistent performance at any scale

Lambda

- Run Code for virtually any type of application or backend service - with zero administration
- Just upload your code as a ZIP file or container image, and Lambda will allocate compute execution power and run your code based on the incoming request or event, for any scale of traffic.
- You can set up your code to automatically trigger from over 200 AWS services and SaaS applications or call it directly from any web or mobile app.
- Language Support for Node.js, Python, Go, Java, and more and use both serverless and container tools, such as AWS SAM or Docker CLI, to build, test, and deploy your functions

Lambda

What is a Lambda function?

- The code you run on AWS Lambda is called a “Lambda function.”
- After you create your Lambda function it is always ready to run as soon as it is triggered, similar to a formula in a spreadsheet.
- Each function includes your code as well as some associated configuration information, including the function name and resource requirements.

Lambda Functions

- Lambda functions are “stateless”, with no affinity to the underlying infrastructure, so that Lambda can rapidly launch as many copies of the function as needed to scale to the rate of incoming events.
- After you upload your code to AWS Lambda, you can associate your function with specific AWS resources (e.g. S3 bucket, Amazon DynamoDB table, Amazon Kinesis stream, or Amazon SNS notification).
- When the resource changes, Lambda will execute your function and manage the compute resources as needed in order to keep up with incoming requests.

Built-in Fault Tolerance

- AWS Lambda maintains compute capacity across multiple Availability Zones in each region to help protect your code against individual machine or data center facility failures.
- Both AWS Lambda and the functions running on the service provide predictable and reliable operational performance.
- AWS Lambda is designed to provide high availability for both the service itself and for the functions it operates. No maintenance windows or scheduled downtimes.

Lambda Concepts

- Function
- Trigger
- Event
- Execution environment
- Deployment package
- Runtime
- Layer
- Extension
- Concurrency
- Qualifier

Lambda Concepts

- Function
  - A function is a resource that you can invoke to run your code in Lambda. A function has code to process the events that you pass into the function or that other AWS services send to the function.
- Trigger
  - A trigger is a resource or configuration that invokes a Lambda function. This includes AWS services that you can configure to invoke a function, applications that you develop, and event source mappings. An event source mapping is a resource in Lambda that reads items from a stream or queue and invokes a function

Lambda Concepts

- Event
  - An event is a JSON-formatted document that contains data for a Lambda function to process. The runtime converts the event to an object and passes it to your function code. When you invoke a function, you determine the structure and contents of the event.
- Execution environment
  - An execution environment provides a secure and isolated runtime environment for your Lambda function. An execution environment manages the processes and resources that are required to run the function. The execution environment provides lifecycle support for the function and for any extensions associated with your function.

Lambda Concepts

- Deployment package
  - You deploy your Lambda function code using a deployment package. Lambda supports two types of deployment packages:
    - A .zip file archive that contains your function code and its dependencies. Lambda provides the operating system and runtime for your function.
    - A container image that is compatible with the Open Container Initiative (OCI) specification. You add your function code and dependencies to the image. You must also include the operating system and a Lambda runtime.

Lambda Concepts

- Runtime
  - The runtime provides a language-specific environment that runs in an execution environment. The runtime relays invocation events, context information, and responses between Lambda and the function. You can use runtimes that Lambda provides, or build your own. If you package your code as a .zip file archive, you must configure your function to use a runtime that matches your programming language. For a container image, you include the runtime when you build the image.
- Layer
  - A Lambda layer is a .zip file archive that can contain additional code or other content. A layer can contain libraries, a custom runtime, data, or configuration files.

Lambda Concepts

- Extension
  - Lambda extensions enable you to augment your functions. For example, you can use extensions to integrate your functions with your preferred monitoring, observability, security, and governance tools.
- Concurrency
  - Concurrency is the number of requests that your function is serving at any given time. When your function is invoked, Lambda provisions an instance of it to process the event. When the function code finishes running, it can handle another request. If the function is invoked again while a request is still being processed, another instance is provisioned, increasing the function's concurrency.

Lambda Concepts

- Qualifier
  - When you invoke or view a function, you can include a qualifier to specify a version or alias. A version is an immutable snapshot of a function's code and configuration that has a numerical qualifier. For example, my-function:1. An alias is a pointer to a version that you can update to map to a different version, or split traffic between two versions. For example, my-function:BLUE. You can use versions and aliases together to provide a stable interface for clients to invoke your function.

Context

```
import time

def lambda_handler(event, context):
    print("Lambda function ARN:", context.invoked_function_arn)
    print("CloudWatch log stream name:", context.log_stream_name)
    print("CloudWatch log group name:",  context.log_group_name)
    print("Lambda Request ID:", context.aws_request_id)
    print("Lambda function memory limits in MB:",
context.memory_limit_in_mb)
    # We have added a 1 second delay so you can see the time remaining
in get_remaining_time_in_millis.
    time.sleep(1)
    print("Lambda time remaining in MS:",
context.get_remaining_time_in_millis())
```

Logging

- Base Logging
  - print()
- Logging Library

```
import os
import logging
logger = logging.getLogger()
logger.setLevel(logging.INFO)
def lambda_handler(event, context):
    logger.info('## ENVIRONMENT VARIABLES')
    logger.info(os.environ) logger.info('## EVENT')
    logger.info(event)
```

Versioning

- Use versions to manage deployment of your functions
- A function version includes the following information:
  - The function code and all associated dependencies.
  - The Lambda runtime that invokes the function.
  - All of the function settings, including the environment variables.
  - A unique Amazon Resource Name (ARN) to identify the specific version of the function.

Invocations

- Invoke using Lambda console, the Lambda API, the AWS SDK, the AWS CLI, and AWS toolkits.
- You can also configure other AWS services to invoke your function, or you can configure Lambda to read from a stream or queue and invoke your function.
- When you invoke a function, you can choose to invoke it synchronously or asynchronously.

Synchronous Vs Asynchronous

- Synchronous invocation
  - Wait for the function to process the event and return a response.
- Asynchronous invocation
  - Lambda queues the event for processing and returns a response immediately. For asynchronous invocation, Lambda handles retries and can send invocation records to a destination.

AWS CLI

- Synchronous
  - aws lambda invoke --function-name my-function --payload '{ "key": "value" }' response.json
- Asynchronous
  - aws lambda invoke --function-name my-function --invocation-type Event --payload '{ "key": "value" }' response.json

Concurrency for a Lambda function

- Two types of Concurrency Controls:
  - Reserved concurrency – Guarantees the maximum number of concurrent instances for the function. When a function has reserved concurrency, no other function can use that concurrency. There is no charge for configuring reserved concurrency for a function.
  - Provisioned concurrency –Initializes a requested number of execution environments so that they are prepared to respond immediately to your function's invocations. Note that configuring provisioned concurrency incurs charges to your AWS account.

Cold Vs Hot Container

- When running a serverless function, it will stay active (a.k.a., hot) as long as you're running it. Your container stays alive, ready and waiting for execution.
- After a period of inactivity, your cloud provider will drop the container, and your function will become inactive, (a.k.a., cold).
- A cold start happens when you execute an inactive function. The delay comes from your cloud provider provisioning your selected runtime container and then running your fn.

Temporary storage with /tmp

- The Lambda execution environment provides a file system for your code to use at /tmp. This space has a fixed size of 512 MB. The same Lambda execution environment may be reused by multiple Lambda invocations to optimize performance. The /tmp area is preserved for the lifetime of the execution environment and provides a transient cache for data between invocations. Each time a new execution environment is created, this area is deleted.

```
import os, zipfile
os.chdir('/tmp')
with zipfile.ZipFile(myzipfile, 'r') as zip:
    zip.extractall()
```

API Gateway

- Amazon API Gateway is a fully managed service that makes it easy for developers to create, publish, maintain, monitor, and secure APIs at any scale.
- APIs act as the "front door" for applications to access data, business logic, or functionality from your backend services.
- Create RESTful APIs and WebSocket APIs that enable real-time two-way communication applications.
- API Gateway supports containerized and serverless workloads & web applications.

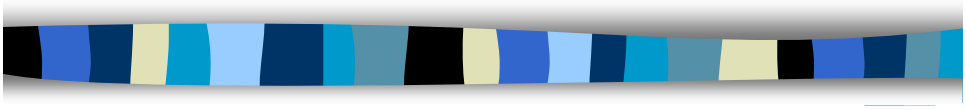
Benefits of API Gateway

- Efficient API development
- Easy Monitoring
- Performance at any scale
- Flexible security controls
- Cost savings at scale
- RESTful API options

Exposing Lambda function as API Gateway

fands™
A Fast AND Steady Approach

Step Functions



fands™

Step Functions

- Serverless function orchestrator that makes it easy to sequence AWS Lambda functions and multiple AWS services into business-critical applications.
- Through its visual interface, you can create and run a series of checkpointed and event-driven workflows that maintain the application state. The output of one step acts as an input to the next.
- Each step in your application executes in order, as defined by your business logic.

Main Features

- Sequencing
- Error Handling
- Retry Logic
- Branching
- Parallel Paths
- Human Interaction
- Retaining state across function calls
- Tracing

Workflow Types

- Step Functions has two workflow types.
- Standard workflows
  - have exactly-once workflow execution and can run for up to one year.
  - Ideal for long-running, auditable workflows, as they show execution history and visual debugging
- Express workflows
  - have at-least-once workflow execution and can run for up to five minutes.
  - Ideal for high-event-rate workloads, such as streaming data processing and IoT data ingestion.

Standard and Express workflows

- Standard workflows
  - 2,000 per second execution rate
  - 4,000 per second state transition rate
  - Priced per state transition
  - Shows execution history and visual debugging
  - Supports all service integrations and patterns
- Express workflows
  - 100,000 per second execution rate
  - Nearly unlimited state transition rate
  - Priced per number and duration of executions
  - Sends execution history to CloudWatch
  - Supports all service integrations and most patterns

Use Cases

- Use case #1: Function orchestration
- Use case #2: Branching
- Use case #3: Error handling
- Use case #4: Human in the loop
- Use case #5: Parallel processing
- Use case #6: Dynamic parallelism

Service Integrations

- Request a response (default)
  - Call a service, and let Step Functions progress to the next state after it gets an HTTP response.
- Run a job (.sync)
  - Call a service, and have Step Functions wait for a job to complete.
- Wait for a callback with a task token (.waitForTaskToken)
  - Call a service with a task token, and have Step Functions wait until the task token returns with a callback.

States

- Individual states can make decisions based on their input, perform actions, and pass output to other states.
- In AWS Step Functions you define your workflows in the Amazon States Language.
- The Step Functions console provides a graphical representation of that state machine to help visualize your application logic.
- States are elements in your state machine. A state is referred to by its name, which can be any string, but which must be unique within the scope of the entire state machine

State Functions

- Task
  - Do some work in your state machine (a Task state)
- Choice
  - Make a choice between branches of execution
- Fail or Succeed
  - Stop an execution with a failure or success
- Pass
  - Simply pass its input to its output or inject some fixed data
- Wait
  - Provide a delay for a certain amount of time or until a specified time/date
- Parallel
  - Begin parallel branches of execution

Pass State

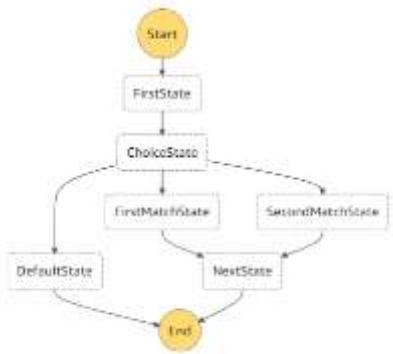
```
{
  "Comment": "A Hello World example for  Pass states",
  "StartAt": "Hello",
 "States": {
    "Hello": {
      "Type": "Pass",
      "Result": "Hello",
      "Next": "World"
    },
    "World": {
      "Type": "Pass",
      "Result": "World",
      "End": true } } }
```

Map – Array Processing

```
{
  "Comment": "..",
  "StartAt": "Map",
  "States": {
    "Map": {
      "Type": "Map",
      "ItemsPath": "$.array",
      "ResultPath": "$.array",
      "MaxConcurrency": 2,
      "Next": "Final State",
      "Iterator": {
        "StartAt": "Pass",
"States": {
        "Pass": {
          "Type": "Pass",
          "Result": "Done!",
          "End": true
        }
      }
    }
  },
  "Final State": {
    "Type": "Pass",
    "End": true
  }
}
```

Choice - Branching

```
"ChoiceState": {
   "Type" : "Choice",
   "Choices": [
   {
    "Variable": "$.foo",
    "NumericEquals": 1,
    "Next": "FirstMatchState"
     },
     {
     "Variable": "$.foo",
     "NumericEquals": 2,
     "Next": "SecondMatchState"
      }
```



Wait State – Timed wait

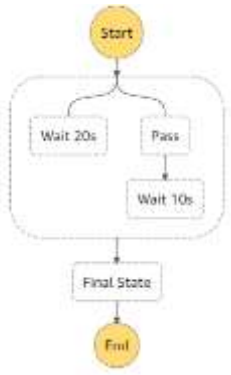
```
"wait_using_seconds": {
    "Type": "Wait",
    "Seconds": 10,
  },

"wait_using_timestamp": {
    "Type": "Wait",
    "Timestamp": "2015-09-04T01:59:00Z",
  },

"wait_using_timestamp_path": {
    "Type": "Wait",
    "TimestampPath": "$.expirydate",
  },
"wait_using_seconds_path": {
    "Type": "Wait",
    "SecondsPath": "$.expiryseconds",
  },
```

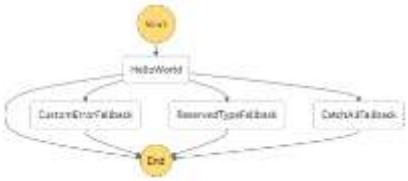
Parallel – Parallel Branches

```
"Parallel": {
    "Type": "Parallel",
    "Next": "Final State",
    "Branches": [
     { "StartAt": "Wait 20s",
       "States": { "Wait 20s": {
          "Type": "Wait",
          "Seconds": 20,
          "End": true     } } },
     { "StartAt": "Pass",
       "States": {
         "Pass": {
          "Type": "Pass",
          "Next": "Wait 10s“    },
        "Wait 10s": {
```



Catch – Handling Errors

```
"HelloWorld": {
 "Type": "Task",
 "Resource": "lambdaArn",
 "Catch": [     {
 "ErrorEquals":["CustomError"],
 "Next": "CustomErrorFallback“
},
 {
"ErrorEquals":["States.TaskFailed"],
 "Next": ReservedTypeFallback"
},
 {    "ErrorEquals": ["States.ALL"],
       "Next": "CatchAllFallback“ }
    ],
      "End": true
```



Periodically Starting

- CloudWatch Events
  - Delivers a near real-time stream of system events that describe changes in AWS resources
- CloudWatch Events
  - Amazon EventBridge is the preferred way to manage your events. CloudWatch Events and EventBridge are the same underlying service and API, but EventBridge provides more features. Changes you make in either CloudWatch or EventBridge will appear in each console

Periodically Starting

- Start state machine execution after every one minute
  - Create State Machine
  - Select State machine
    - Actions
    - Create event bridge(CloudWatch) rule
    - After every 1 minute
  - Check
    - State Machine Executions

Exposing as API Gateway

- You can use Amazon API Gateway to associate your AWS Step Functions APIs with methods in an API Gateway API. When an HTTPS request is sent to an API method, API Gateway invokes your Step Functions API actions.
- Start a Step Functions execution by calling StartExecution and DescribeExecution to get the result.

Amazon S3 Events as trigger

- Amazon EventBridge to execute an AWS Step Functions state machine in response to an event or on a schedule.

Activities

- Activities are an AWS Step Functions feature that enables you to have a task in your state machine where the work is performed by a worker that can be hosted on Amazon Elastic Compute Cloud (Amazon EC2), Amazon Elastic Container Service (Amazon ECS), AWS Lambda, mobile devices—basically anywhere.

Overview

- In AWS Step Functions, activities are a way to associate code running somewhere (known as an activity worker) with a specific task in a state machine.
- You can create an activity using the Step Functions console, or by calling CreateActivity. This provides an Amazon Resource Name (ARN) for your task state. Use this ARN to poll the task state for work in your activity worker.

Human Interaction

- The most important feature of creating workflows with Step Functions is the ability to control its execution with external input. We can pause, continue, or stop the workflows whenever we want. This is especially important when we need to have human interactions with the workflow.

Serverless Manual Approval

https://aws.amazon.com/blogs/compute/implementing-serverless-manual-approval-steps-in-aws-step-functions-and-amazon-api-gateway/

Maintaining

- Logging and monitoring
  - Cloud Watch
    - Metrics
    - Alarms
  - EventBridge Events
  - AWS CloudTrail
- Logging using CloudWatch Logs
  - X-Ray

X-Ray

- Use AWS X-Ray to visualize the components of your state machine, identify performance bottlenecks, and troubleshoot requests that resulted in an error.
- State machine sends trace data to X-Ray, and X-Ray processes the data to generate a service map and searchable trace summaries.

X-Ray

- This gives you a detailed overview of an entire Step Functions request. Step Functions will send traces to X-Ray for state machine executions, even when a trace ID is not passed by an upstream service. You can use an X-Ray service map to view the latency of a request, including any AWS services that are integrated with X-Ray. You can also configure sampling rules to tell X-Ray which requests to record, and at what sampling rates, according to criteria that you specify.

AWS SAM

- Install SAM CLI
- Download, build, and deploy a sample AWS SAM application that contains an AWS Step Functions state machine. This application creates a mock stock trading workflow which runs on a pre-defined schedule

QUESTION / ANSWERS

THANKING YOU !

www.fandsindia.com