**1. Conhecendo o AWS Step Functions**

**O que é:**  
O AWS Step Functions é um serviço da AWS que permite **orquestrar vários serviços da AWS em fluxos de trabalho visuais**. Ele ajuda a coordenar tarefas de forma **automatizada, sequencial ou paralela**, sem precisar escrever muito código para controlar falhas ou dependências.

**Principais conceitos:**

* **State Machine (Máquina de Estados):**  
  É o núcleo do Step Functions. Define **como as tarefas e decisões se conectam** no fluxo de trabalho. Cada “estado” representa uma ação ou decisão.
* **Estados principais:**
  + **Task (Tarefa):** Executa uma função, como uma Lambda, ECS ou chamada a outro serviço.
  + **Choice (Escolha):** Permite criar **condições** para diferentes caminhos no fluxo.
  + **Parallel (Paralelo):** Executa várias tarefas ao mesmo tempo.
  + **Wait (Aguardar):** Pausa o fluxo por um período definido.
  + **Fail / Succeed:** Finalizam o fluxo, indicando sucesso ou falha.
* **Fluxo de execução:**
  + O Step Functions inicia o workflow.
  + Passa por cada estado conforme definido na máquina de estados.
  + Registra logs e rastreia cada execução.
  + Pode lidar com **retries (tentativas) e erros automaticamente**.

**Vantagens:**

* Visual e fácil de monitorar com **console gráfico**.
* Coordena **serviços AWS diversos** sem escrever scripts complexos.
* **Alta confiabilidade:** pode lidar automaticamente com erros e falhas.
* Permite **integração com Lambda, SQS, SNS, ECS, DynamoDB, entre outros**.
* Ideal para **processos de negócios, ETL, automações complexas e pipelines de dados**.

**Exemplo de uso:**

* Um e-commerce pode usar Step Functions para:
  1. Receber um pedido.
  2. Validar pagamento (Lambda).
  3. Atualizar estoque (DynamoDB).
  4. Enviar notificação de confirmação (SNS).
  5. Finalizar o processo.

**2 - Benefícios e Projeto Modelo no AWS Step Functions**

**Benefícios do AWS Step Functions:**

1. **Orquestração de Serviços:**
   * Coordena vários serviços AWS de forma visual e organizada.
   * Elimina a necessidade de scripts complexos para controlar a sequência de tarefas.
2. **Resiliência e Tolerância a Falhas:**
   * Possui **retry automático**, tratamento de erros e escalonamento de falhas.
   * Mantém o histórico das execuções para auditoria e depuração.
3. **Monitoramento e Visibilidade:**
   * Painel gráfico no console AWS que mostra **status de cada execução em tempo real**.
   * Logs integrados ao CloudWatch para análise detalhada.
4. **Automação de Processos Complexos:**
   * Suporta **decisões condicionais**, paralelismo e temporizações entre tarefas.
   * Ideal para pipelines de dados, processamento de pedidos ou fluxos de negócios.
5. **Flexibilidade:**
   * Integra facilmente com **Lambda, ECS, SNS, SQS, DynamoDB**, entre outros.
   * Permite adaptar o fluxo sem precisar alterar cada serviço individualmente.

**Projeto Modelo (Best Practice / Exemplo de Arquitetura):**

* **Cenário:** Processamento de pedidos de um e-commerce.
* **Passo a passo do fluxo modelo:**
  1. **Início (Start):** Recebe o evento do pedido.
  2. **Validação do Pedido (Task):** Lambda valida estoque e informações.
  3. **Processamento de Pagamento (Task):** Lambda ou integração com sistema externo.
  4. **Decisão (Choice):**
     + Se pagamento aprovado → continuar processamento.
     + Se pagamento recusado → enviar notificação e finalizar fluxo.
  5. **Atualização de Estoque (Task):** DynamoDB ou outro banco.
  6. **Notificação ao Cliente (Task):** SNS ou Lambda envia email/SMS.
  7. **Encerramento (Succeed / Fail):** Finaliza o workflow.

**Dicas de Projeto Modelo:**

* Sempre definir **trilhas de erro (error handling)** para cada Task.
* Usar **Wait** quando precisar de delays (ex.: verificar status de pagamento).
* Modularizar tarefas em funções Lambda pequenas e específicas.
* Registrar logs detalhados no CloudWatch para monitoramento.

**3 - Realizando Validações no AWS Step Functions**

**O que são validações no Step Functions:**

* Validações garantem que **os dados e condições atendam aos requisitos antes de prosseguir no fluxo**.
* Permitem **evitar erros e decisões incorretas** durante a execução da máquina de estados.

**Principais formas de validação:**

1. **Validação com Choice State (Estado de Escolha):**
   * Permite criar **condições lógicas** para decidir caminhos diferentes no workflow.
   * Pode verificar valores, tipos e presença de atributos nos dados de entrada.
   * Exemplo:
     + Se statusPagamento == “aprovado” → prosseguir.
     + Se statusPagamento == “recusado” → encerrar fluxo ou enviar notificação.
2. **Validação de Entrada (InputPath / Parameters / ResultPath):**
   * **InputPath:** seleciona quais partes do input passar para o estado.
   * **Parameters:** mapeia ou transforma dados antes da execução do estado.
   * **ResultPath:** define onde o resultado de um estado será armazenado para uso nos próximos estados.
3. **Validação de Saída (OutputPath):**
   * Filtra ou transforma o resultado do estado antes de enviá-lo para o próximo estado.
   * Ajuda a garantir que **apenas os dados esperados sigam no fluxo**.
4. **Tratamento de Erros e Fallbacks (Catch / Retry):**
   * **Retry:** tenta novamente em caso de falha temporária (ex.: falha de rede).
   * **Catch:** define ações alternativas quando ocorre um erro (ex.: registrar log, enviar alerta).

**Boas práticas de validação:**

* Sempre validar **inputs e outputs** para evitar propagação de dados incorretos.
* Usar **Choice State** para decisões críticas no workflow.
* Combinar **Retry e Catch** para aumentar resiliência.
* Documentar **as condições de cada Choice** para facilitar manutenção.

**4 - Criando e Executando Lambda no AWS Step Functions**

**Integração com AWS Lambda:**

* O Step Functions permite **orquestrar funções Lambda** como parte de um fluxo de trabalho.
* Cada **Task State** pode chamar uma Lambda para executar uma ação específica.

**Passos para criar e executar Lambda no Step Functions:**

1. **Criar a função Lambda:**
   * Escolher runtime (Python, Node.js, Java, etc.).
   * Escrever a lógica da função.
   * Testar a função isoladamente para garantir que funciona corretamente.
2. **Definir a Task State no Step Functions:**
   * Criar uma **máquina de estados** no console ou via JSON/YAML (ASL – Amazon States Language).
   * Adicionar um **Task State** que aponte para a Lambda criada.
   * Configurar **InputPath, Parameters e ResultPath** se precisar manipular dados.
3. **Executar o workflow:**
   * Iniciar a execução da máquina de estados no console AWS.
   * O Step Functions chama a Lambda automaticamente.
   * Monitorar status e logs no console e no **CloudWatch**.

**Exemplo de fluxo simples com Lambda:**

1. Start → Task (Lambda: valida pedido) → Choice (aprovado/recusado)
2. Se aprovado → Task (Lambda: processa pagamento) → Task (Lambda: envia notificação) → Succeed
3. Se recusado → Task (Lambda: envia alerta) → Fail

**Boas práticas ao integrar Lambda:**

* Manter funções **simples e modulares**.
* Tratar erros dentro da Lambda e usar **Retry/Catch** no Step Functions para maior resiliência.
* Monitorar logs no **CloudWatch** para rastrear falhas ou dados inesperados.
* Limitar tempo de execução da Lambda dentro do esperado pelo Step Functions.