Serverless Workflow Orchestration on AWS

Hello!

I am Bharat Vishal Tiwary

SDE II @Amazon

LinkedIn: bharattiwary

Article:
https://www.techtimes.com/articles/308145/20241105/techniques-tools-orchestrating-workflows-using-microservices.htm

Serverless Workflow Orchestration on AWS

By orchestrating services, businesses can unlock agility, quickly adapt to changing customer needs, and deliver innovative solutions faster than ever before.

Today's Agenda

- Concepts
- Orchestration in AWS
- Best practices

1

Concepts

What's Serverless Workflow Orchestration?

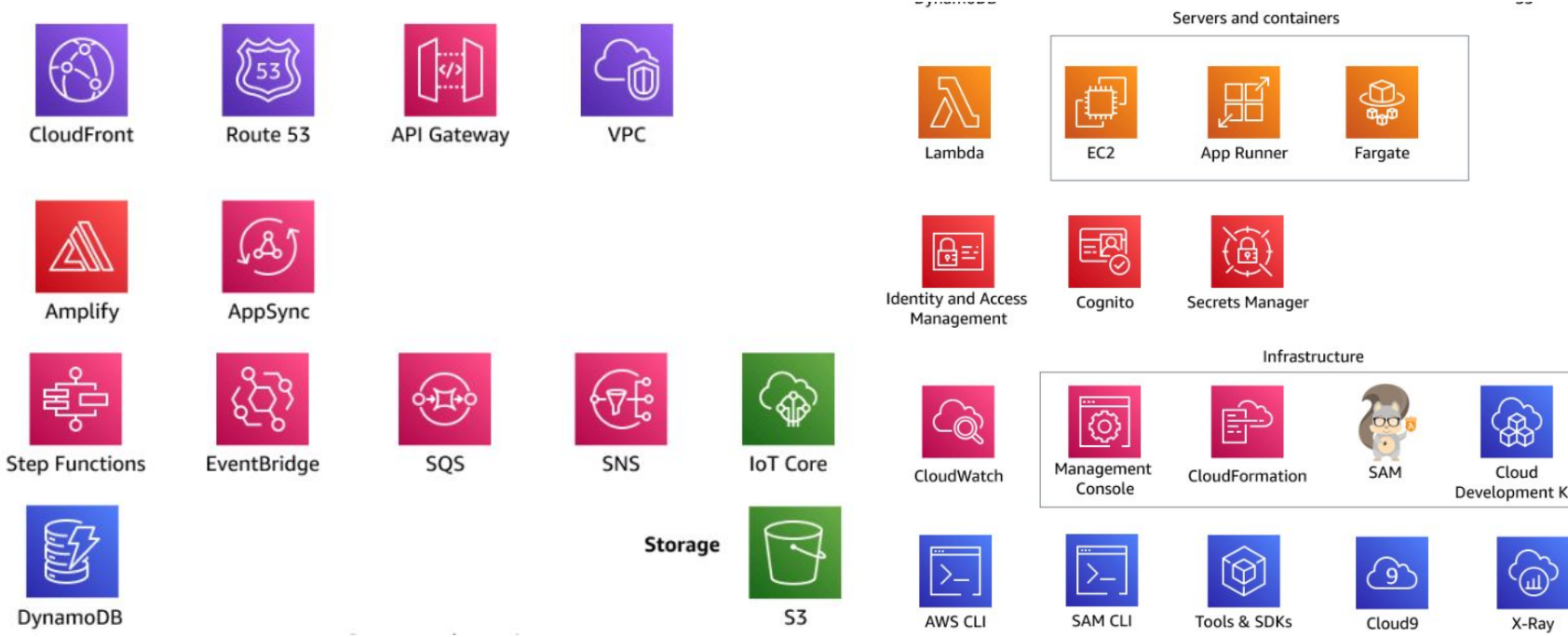
Serverless

Build and run applications without thinking about servers.

What & Why Serverless?

- Serverless computing is a cloud computing execution model where the cloud provider dynamically manages all server resources.
- Physical servers are still used, but they are abstracted away from developers .
- Main value proposition is of focusing on business outcomes while abstracting away the mechanics of computing
- Benefits include: Pay-per-use, Automatic scaling, Reduced operation and infrastructure cost, Faster time to market

AWS Serverless services



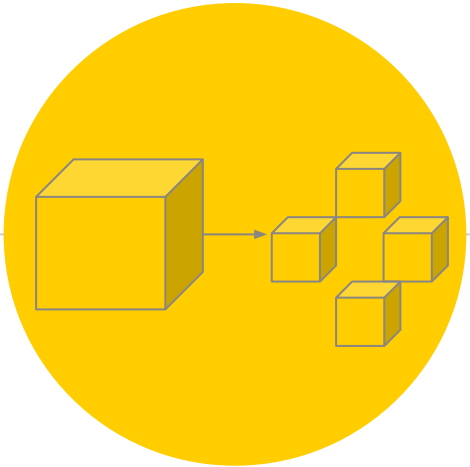
Source: https://docs.aws.amazon.com/serverless/latest/devguide/serverless-core-services.html

Workflow

A workflow is a sequence of tasks that are part of a larger process or goal.

What is a Workflow?

- A workflow is a series of actions that accomplish a particular task, serving as a fundamental unit of work
- Workflows are designed to simplify and automate tasks by combining multiple actions into a coherent sequence .
- In various contexts, workflows serve different purposes



Microservices

Software is composed of small independent services that communicate over well-defined APIs.

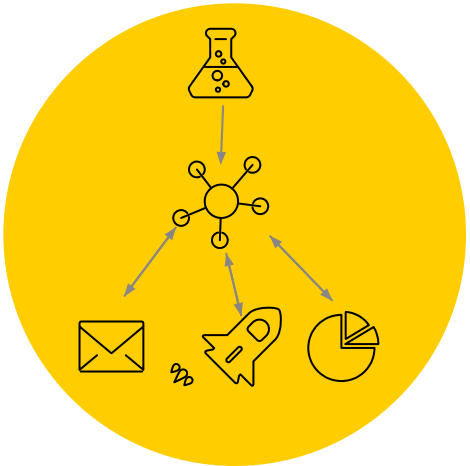
What are Microservices?

- A software architectural approach that structures applications as a collection of small, independent services that communicate over well-defined APIs
- Each service runs in its own process and focuses on doing one thing well, making them simple and granular
- Key Characteristics: Autonomous operation, Technology diversity, Independent Databases
- Teams can operate independently using the "You build it, you run it" DevOps model

Orchestration vs Choreography

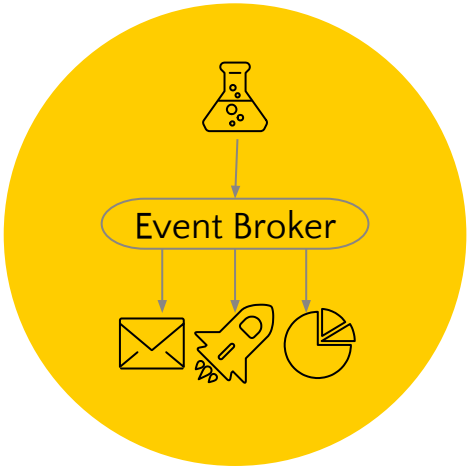
Orchestration

- A central service acts as a brain to coordinate logic.



Choreography

- Each service acts autonomously, also known as Event Driven Design (EDD)



Orchestration vs Choreography

Orchestration

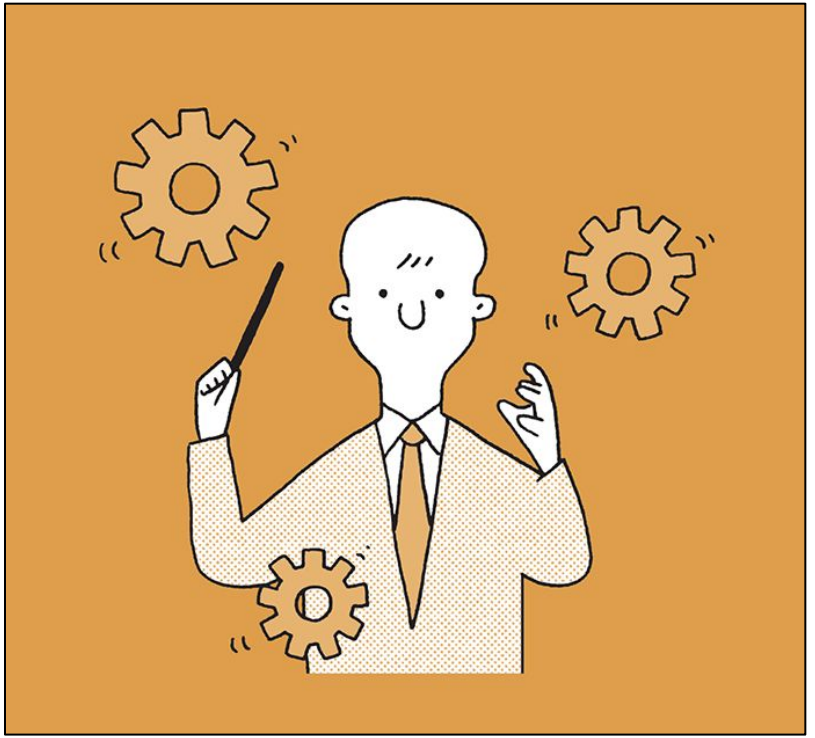
- Explicit and managed by the orchestrator.
- Direct communication via the orchestrator.
- Simpler for defining workflows, more complex orchestration.
- Centralized error handling.
- Less flexible due to central control.

Choreography

- Implicit and managed by individual services.
- Event-based, peer-to-peer communication.
- More complex interactions, simpler service autonomy.
- Distributed error handling.
- Highly flexible and adaptable.

Orchestration use case

- IT service requests
- Compliance
- Invoice processing
- Employee onboarding/offboarding.
- Software deployment



2 Orchestration in AWS

AWS Step Functions

Top Orchestration tools

Apache Airflow

Apache Airflow is an open-source tool for scheduling and monitoring workflows. Developed by Airbnb, it uses directed acyclic graphs (DAGs) to manage complex data pipelines effectively.

AWS Step Functions

AWS Step Functions is a serverless orchestration service that lets you combine AWS services to build to scalable, distributed applications using state machines.

Google Workflows

Google Workflows is a powerful orchestration service from Google Cloud that automates tasks across Google services, boosting productivity and efficiency!

Microsoft Power Automate

Previously known as Microsoft Flow, this tool is a crucial component of the Microsoft ecosystem designed to assist businesses in automating workflows and enhancing productivity across various applications and services.

Dagster

Dagster is an open-source data orchestrator that helps build, run, and monitor data pipelines. It provides a productive environment for data engineers and scientists to define, test, and execute data workflows efficiently.

Argo

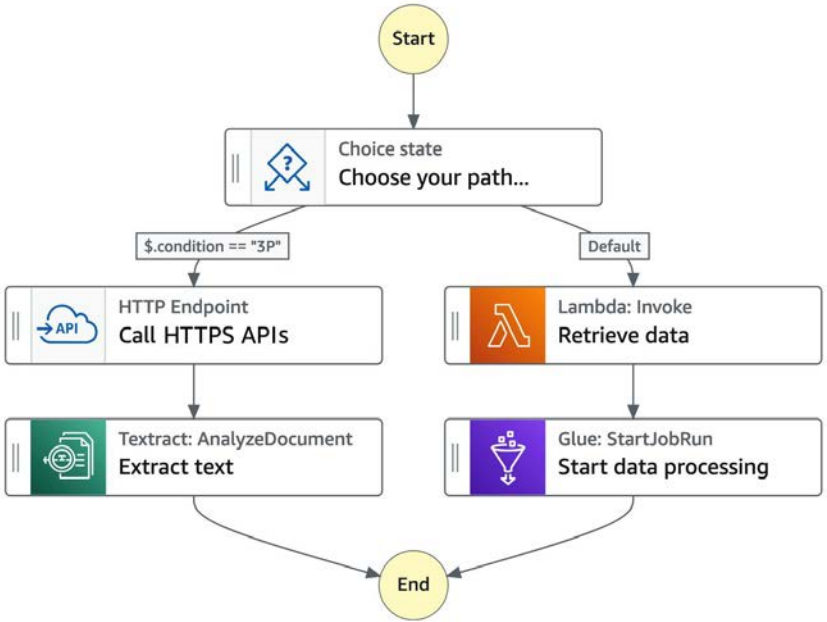
Argo is an open-source workflow engine for orchestrating parallel jobs on Kubernetes. It effectively manages complex dependencies and features advanced scheduling capabilities, making it ideal for large-scale distributed environments.

Why Step functions?

- Low Code / No Code: AWS step function workflows can be created using the workflow studio which is easy to use for non coders.
- Highly Scalable:This solution can easily scale to meet the demands of enterprise-level applications and workflows.
- Reliability: Built on the dependable infrastructure of AWS, it provides high availability and fault tolerance for orchestrated workflows.
- Flexibility: Developers can create workflow logic using familiar programming patterns and seamlessly integrate it with various AWS tools and services.
- Cost-Effective: By Carefully choosing the type of Statefunction Workflow it can be pretty cost effective
- Developers can use CDK for deployment, and the system includes linting tools and CloudFormation validation before deployment ..

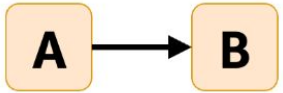
AWS Step Functions

- create workflows to build distributed applications, automate processes, orchestrate microservices, and create data and machine learning pipelines.
- **Key Concepts:** Executions, Tasks, Activities
- In the Step Functions' console, you can visualize, edit, and debug your application's workflow. You can examine the state of each step in your workflow to make sure that your application runs in order and as expected.

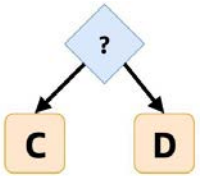


Components of Workflow

Request Response



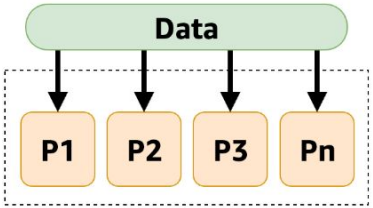
Choose tasks



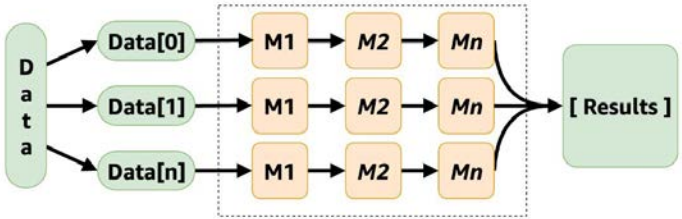
Retry Tasks

Add Human in Loop

Process Data in Parallel

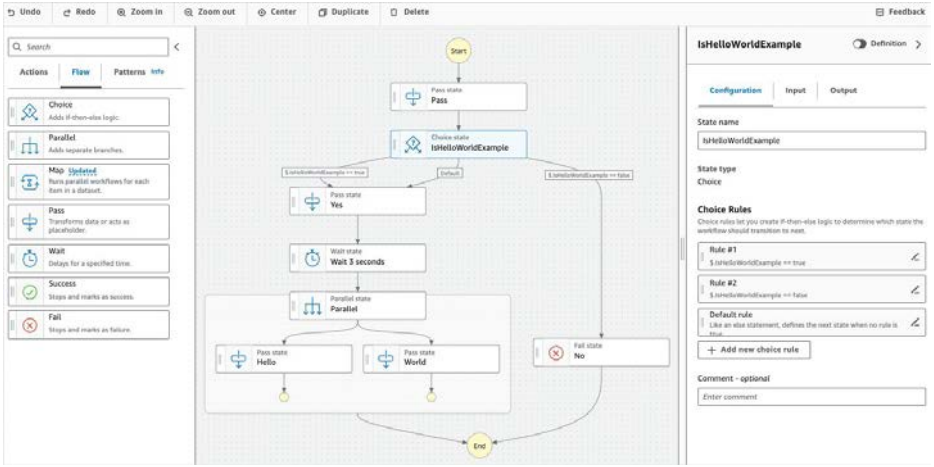


Dynamically Process with Map



Developing Step function

Workflow Studio



Amazon State Language

```
{
  "Comment": "An example of the Amazon States Language using a choice state.",
  "QueryLanguage": "JSONata",
  "StartAt": "FirstState",
  "States": {
    "FirstState": {
      "Type": "Task",
      "Assign": {
        "foo": "{% $states.input.foo_input %}"
      },
      "Resource": "arn:aws:lambda:us-east-1:123456789012:function:FUNCTION_NAME",
      "Next": "ChoiceState"
    },
    "ChoiceState": {
      "Type": "Choice",
      "Default": "DefaultState",
      "Choices": [
        {
          "Next": "FirstMatchState",
          "Condition": "{% $foo = 1 %}"
        },
        {
          "Next": "SecondMatchState",
          "Condition": "{% $foo = 2 %}"
        }
      ]
    },
    "FirstMatchState": {
      "Type": "Task",
      "Resource": "arn:aws:lambda:us-east-1:123456789012:function:OnFirstMatch",
      "Next": "NextState"
    },

    "SecondMatchState": {
      "Type": "Task",
      "Resource": "arn:aws:lambda:us-east-1:123456789012:function:OnSecondMatch",
      "Next": "NextState"
    },

    "DefaultState": {
      "Type": "Fail",
      "Error": "DefaultStateError",
      "Cause": "No Matches!"
```

AWS Step Functions: Use case

- **Orchestrate Microservices**: allowing break down of complex applications into smaller, independent services that can be developed, tested, and deployed independently.
- **Data processing**: Step Functions can be used to process large volumes of data or perform tasks that need to be run periodically.
- **Machine Learning**: Step Functions can be used to build and manage data pipelines, allowing you to move data between different sources and destinations in a reliable and scalable manner.
- **Event-driven architectures**: Step Functions can be used to build event-driven architectures

3 Best Practices

3.1 Design for Scale and Performance

3.1.1 Standard vs Express Workflows

Standard

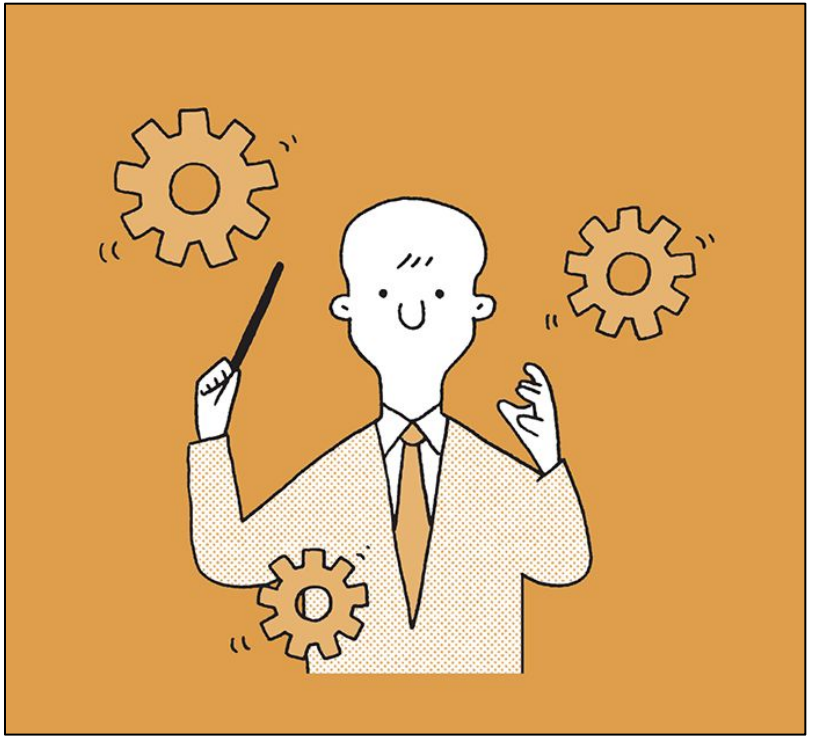
- Can run upto 1 year
- Exactly once execution
- Charged on number of state transitions
- Better suited for non idempotent, long running workflows.

Express

- Limited to 5 minutes max
- At least once execution(async) , At most once (sync)
- $1 for per million executions
- High volume processing workloads

Or Both?

- Standard workflows can act as parent workflows to invoke Express workflows synchronously
- Combines the strengths of both workflow types
- Reliable workflows while maintaining cost efficiency and performance optimization



3.1.2 Right Service Integration

- Consider Lambda for for large number of tasks that can be processed concurrently
- All Lambda functions in Step Functions must be designed to be idempotent
- Lambda function names should not be specified explicitly
- Version control is crucial for both Lambdas and Step Function definitions
- For DynamoDB interactions, use Optimistic Locking/Transactions/Conditional Write to handle race conditions .

3.1.3 Beware of Timeouts

- Amazon States Language doesn't specify timeouts for state machine definitions
- For callback with a task token (.waitForTaskToken), use heartbeats and add the HeartbeatSeconds field in Task state definition.

3.1.4 Retries & Error Handling

- Exceptions should be categorized into RetryableException (like SQS dependency exceptions) and NonRetryableException (like NullPointerException) to simplify the step function graph
- When configuring dependencies, always set timeout and retry policies, especially when connecting to other services like CloudWatch .
- Proactively handle transient lambda exceptions in your state machine to Retry invoking your Lambda function, or to Catch the error.

3.1.5 Monitor & Optimize

Use the AWS CloudWatch service to monitor the performance of your Step Functions workflows. This will help you to identify any bottlenecks or issues that may be impacting performance and allow you to take corrective action as needed.

3

Best Practices

3.2 Security Best Practices

3.2 Security Best Practices

- Use IAM roles for tasks
- Encrypt sensitive data
- Use CloudTrail to monitor Step Functions
- Use resource-level permissions
- Enable CloudWatch logging

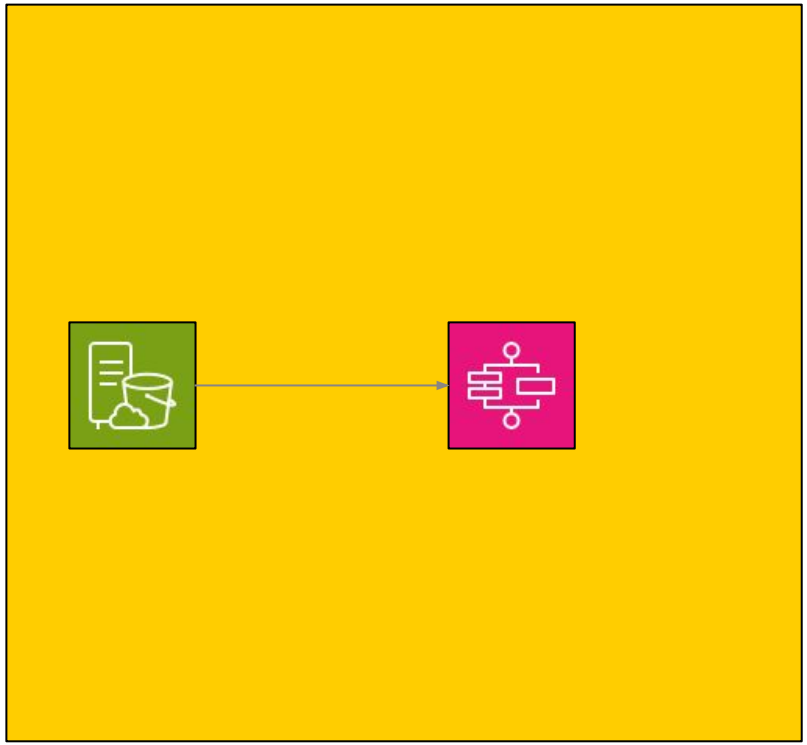
3

Best Practices

3.3 Operation Excellence Best Practices

3.3.1 Passing Large Loads

- Amazon S3 ARNs instead of passing large payloads in Step Functions



3.3.2 CloudWatch For Monitoring

Key Metrics:

- State Transitions
- Throttled State Transitions
- Execution Duration
- Throttled Execution Starts
- Task Failures

3

Best Practices

3.4 Reliability Best Practices

3.4 Reliability Best Practices

- Handle timeouts gracefully
- Beware of event history quota
- Use retries and error handling
- Use Idempotent tasks
- Use Cloudwatch Alarms
- Use CloudTrail logging
- Test your workflows

3

Best Practices

3.4 Cost Optimization Best Practices

3.4 Cost Optimization Best Practices

- Standard vs Express Workflows
- Monitor and Optimize usage
- Use Tagging for Cost Allocation

Thanks!

You can *find me* @

- LinkedIn: bharattiwary
- bharatbvt@gmail.com