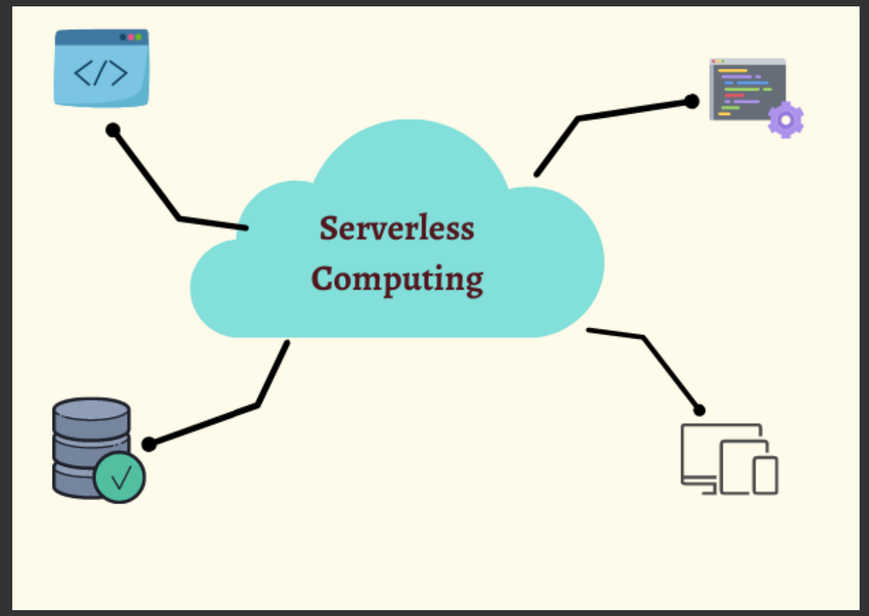
TECH TITANS CMPE 272

Insights into Serverless Computing and AWS Lambda

Exploring the Technical Depths of Cloud Simplification



Serverless Computing

CLOUD COMPUTING PARADIGM SHIFTING FROM SERVER MANAGEMENT TO SERVER ABSTRACTION

- **No Servers to Manage**: Abstraction of physical and virtual servers.
- **Automatic Scaling**: Scales with incoming requests and events.
- **Pay for Value**: Charges based on resource utilization, not idle time.
- **Built-in Fault Tolerance**: High availability across AWS regions and zones.

Greater agility **Less overhead** **Increased scale** **More flexibility** **Faster time to market**

Deep Dive into AWS Lambda

SERVERLESS COMPUTE SERVICE EXECUTING CODE IN RESPONSE TO EVENTS

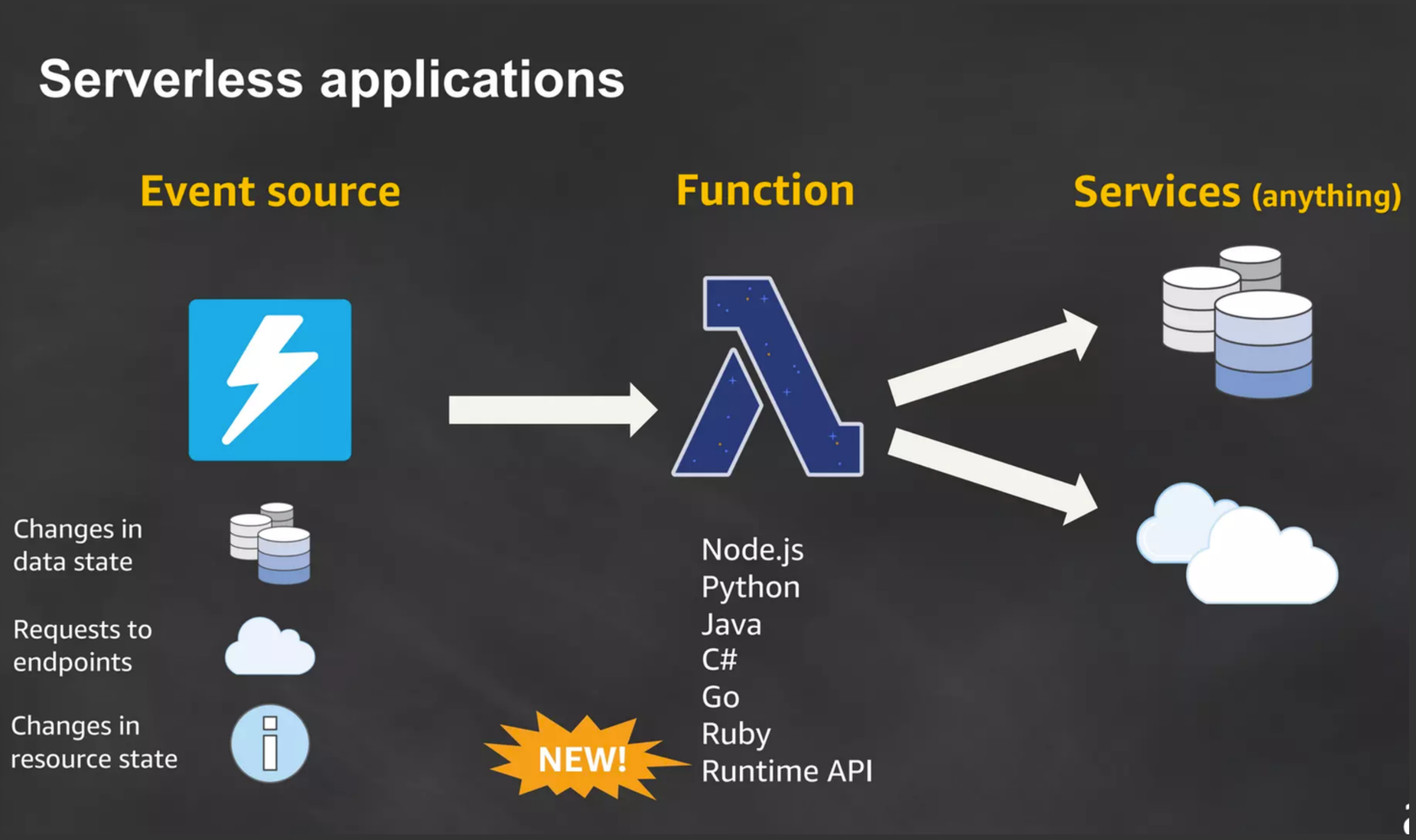
- Event-Triggered Execution: Responds to changes in the environment.
- Stateless Processing: Independent, parallel execution of functions.
- Resource Abstraction: Hides layers of infrastructure complexity.

Load Balancing **Auto Scaling**

Handling Failures **Security Isolation**

OS Management **Managing Utilisation**

Architectural Anatomy of AWS Lambda



Event Sources: Triggers like API Gateway, S3 events.

Lambda Function: Core of execution, language support (Python, Node.js, etc.).

Execution Roles: IAM roles defining permissions and access.

Architectural Anatomy of AWS Lambda

Anatomy of a Lambda function

Handler() function
Function to be executed upon invocation

Event object
Data sent during Lambda function Invocation

Context object
Methods available to interact with runtime information (request ID, log group, more)

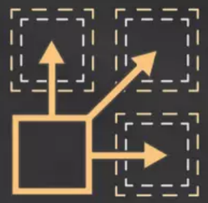
```
public String handleRequest(Book book, Context context) {
    saveBook(book);

    return book.getName() + " saved!";
}
```

AWS Lambda runtime API and layers

FEATURES THAT ALLOW DEVELOPERS TO SHARE , DISCOVER AND DEPLOY BOTH LIBRARIES AND LANGUAGES AS PART OF THEIR SERVERLESS APPLICATIONS

Runtime API enables developers to use Lambda with any programming language.

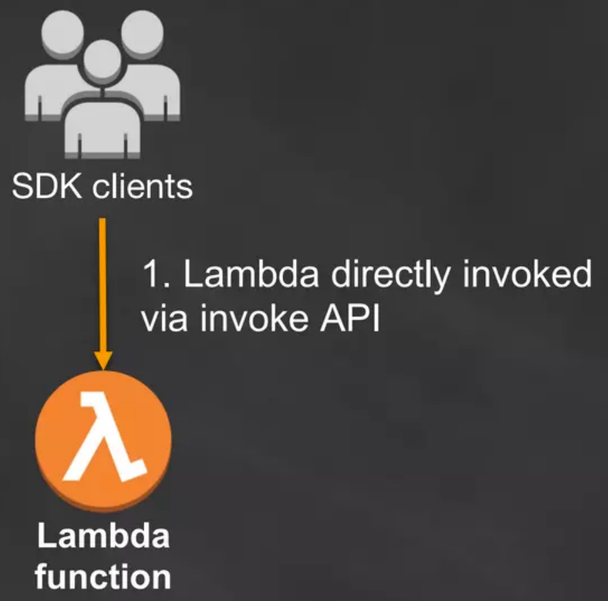


Layers let functions easily share code. Upload layer once, reference within any function.

Layers promote separation of responsibilities, lets developers focus on writing business logic.

Combined, runtime API and layers allow developers to share any programming language or language version with others

Lambda API



API provided by the Lambda service

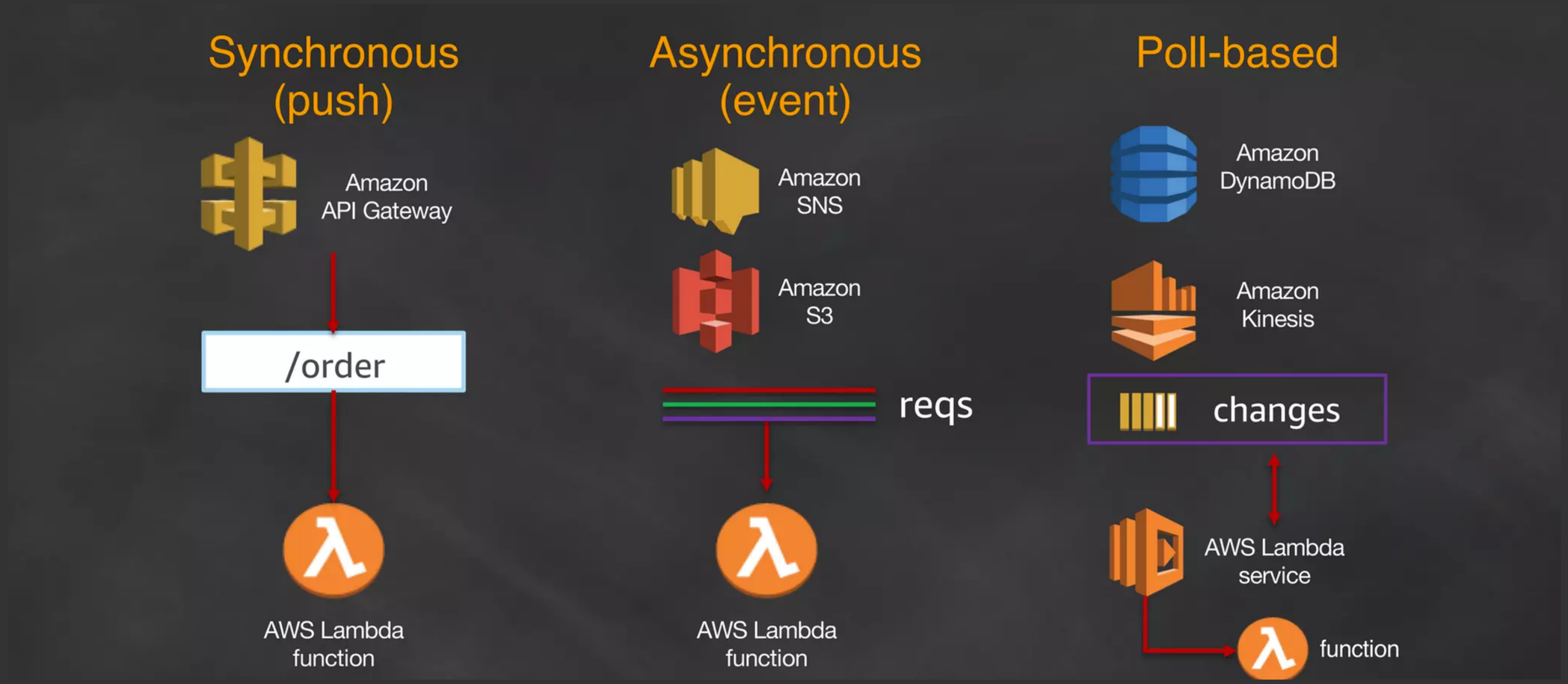
Used by all other services that invoke Lambda across all models

Supports sync and async

Can pass any event payload structure you want

Client included in every SDK

Lambda execution model



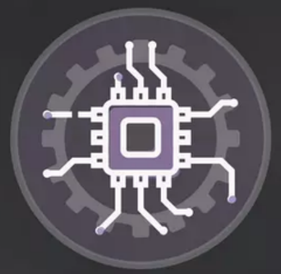
Lambda Use Cases

Web Applications

- Static websites
- Complex web apps
- Packages for Flask and Express

Backends

- Apps & services
- Mobile
- IoT



Data Processing

- Real time
- MapReduce
- Batch

Chatbots

- Powering chatbot logic

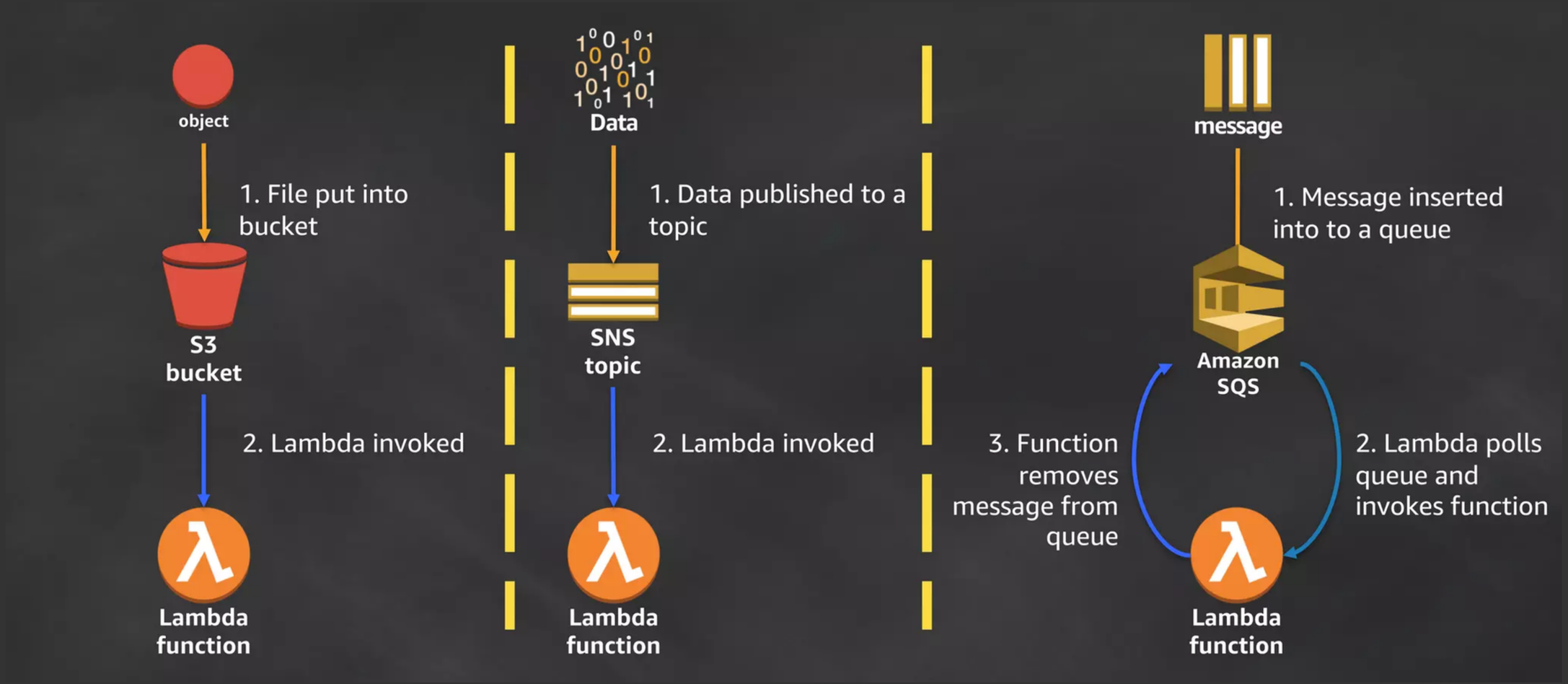
Amazon Alexa

- Powering voice-enabled apps
- Alexa Skills Kit

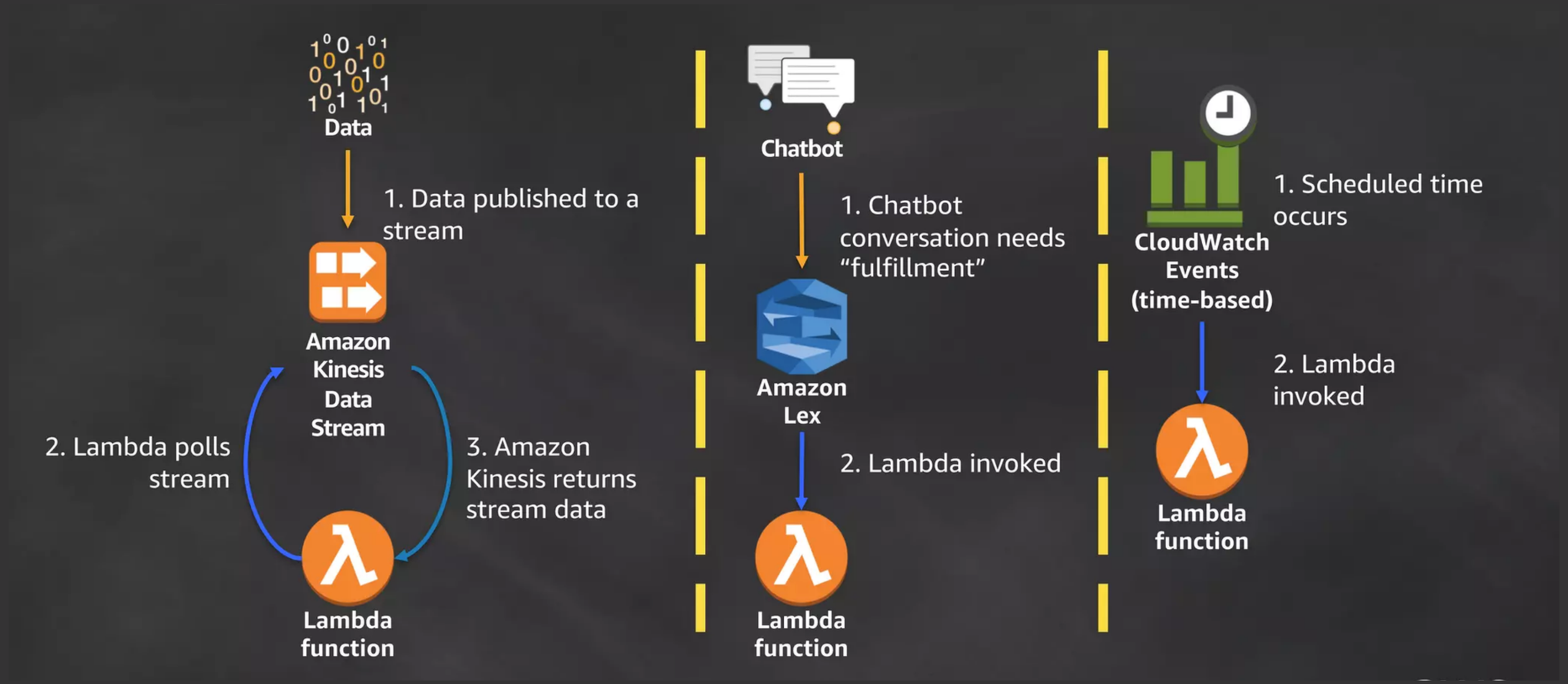
IT Automation

- Policy engines
- Extending AWS services
- Infrastructure management

Serverless Architecture

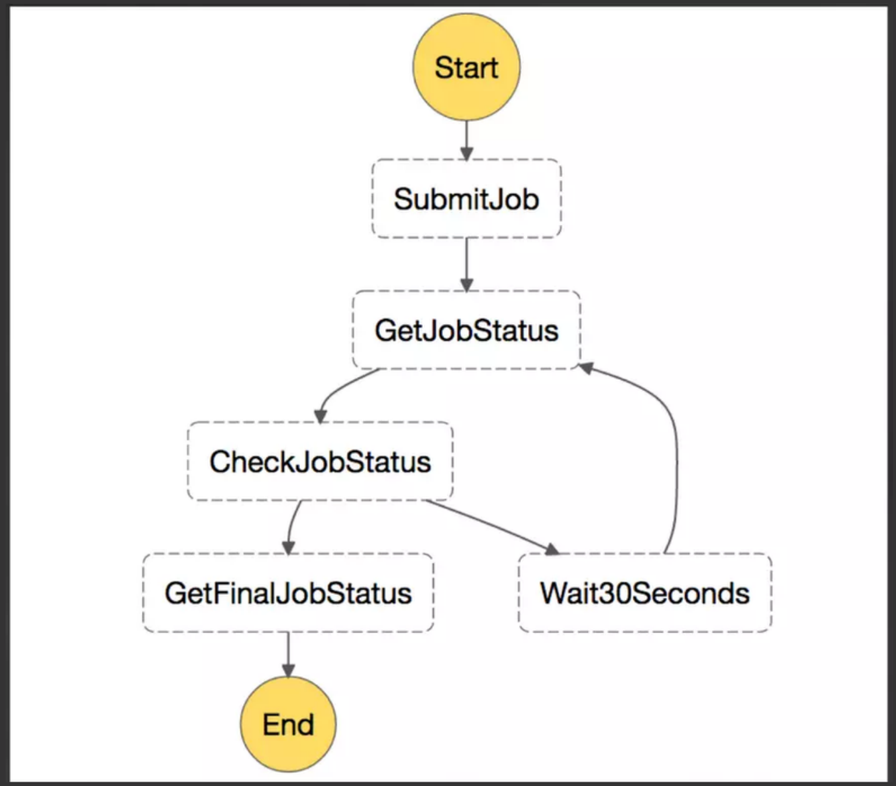


Serverless Architecture



Keep orcherstartion out of code

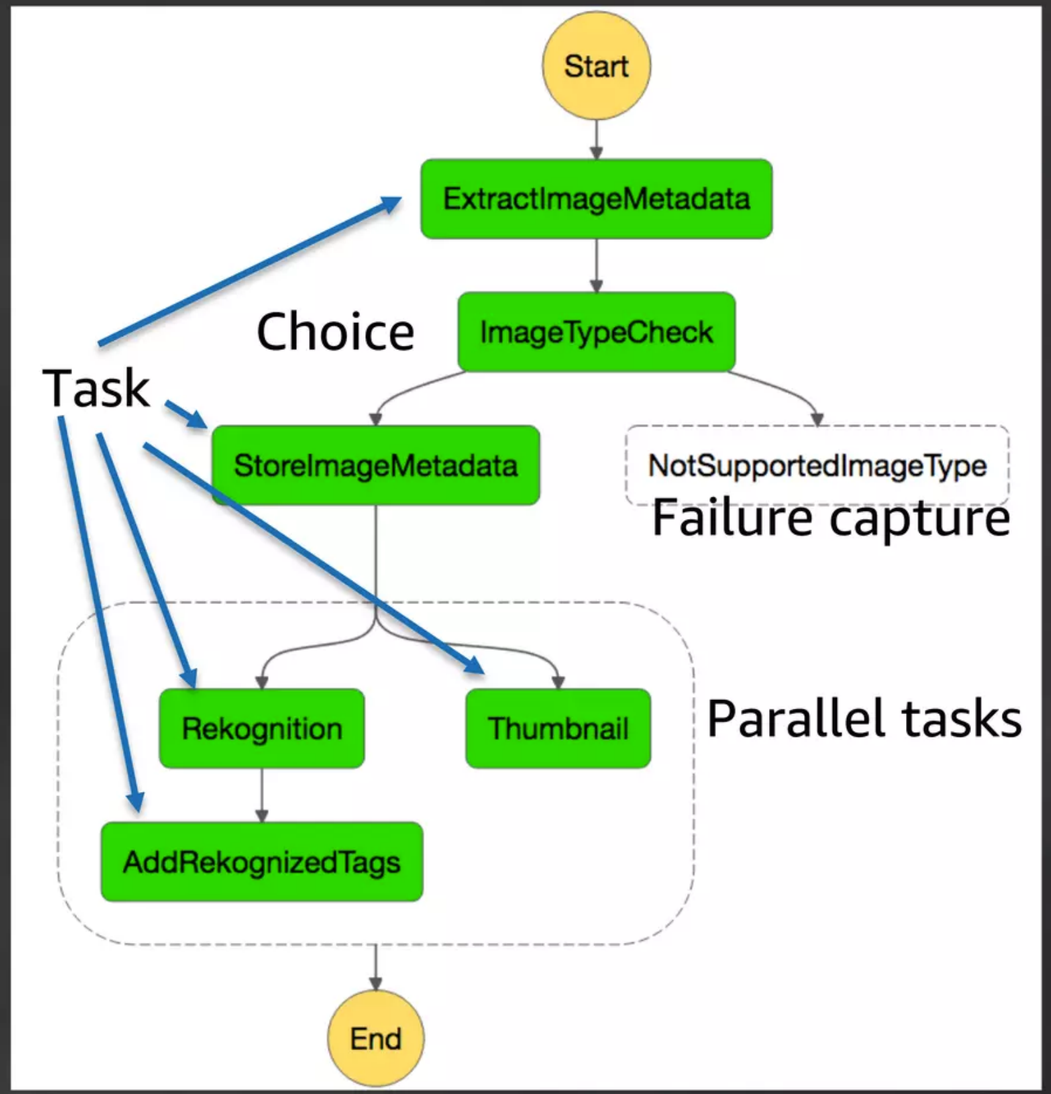
- Separating application logic from workflow management.
- Simplifies code, making it more maintainable and scalable.
- Leverage AWS Step Functions for managing workflows and processes.
- Enhances application reliability and facilitates easier updates and changes.



AWS Step Functions

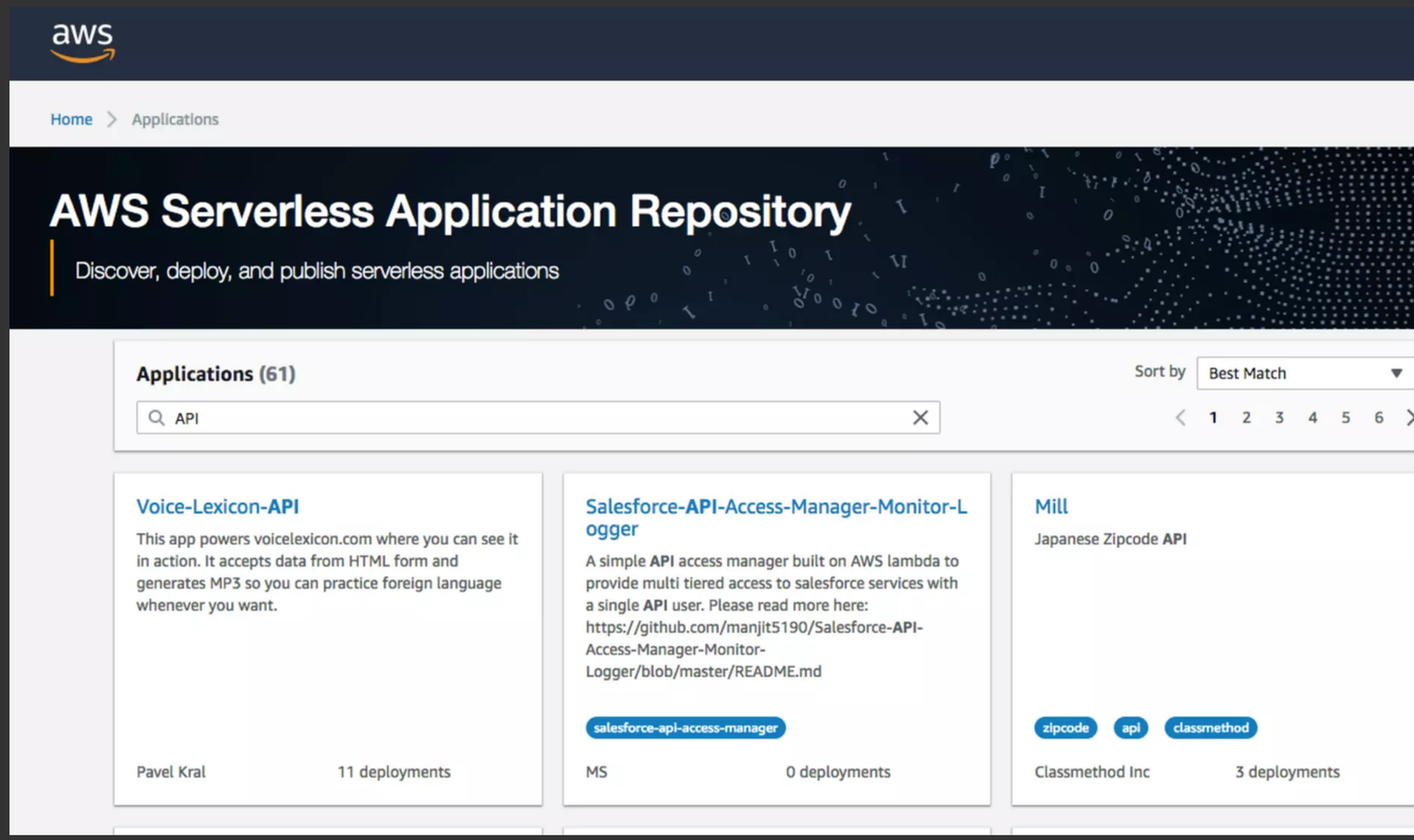
"Serverless" workflow management with zero administration

- Makes it easy to coordinate the components of distributed applications and microservices using visual workflows
- Automatically triggers and tracks each step and retries when there are errors, so your application executes in order and as expected
- Logs the state of each step, so when things do go wrong, you can diagnose and debug problems quickly



AWS Serverless Application Repository

- Managed Repository: A central hub for finding and deploying serverless applications.
- Diverse Applications: Includes solutions for web backends, data processing, IoT, and more.
- Community and Vendor Contributions: Offers access to a wide range of community-driven and vendor-developed applications.
- Seamless AWS Integration: Enables easy deployment and integration with AWS services like Lambda and Amazon S3.



Start with a Framework

- Framework Benefits: Streamlines development, enforces best practices, and provides structured deployment.
- Popular Frameworks: Serverless Framework, AWS SAM (Serverless Application Model), and others tailored to specific languages or needs.
- Rapid Prototyping: Frameworks enable quicker iteration and deployment of serverless applications.

AWS:

AWS SAM

AWS Amplify

aws

AWS

Third-party:

APEX

architect

Claudia.js

Serverless Framework

SPARTA

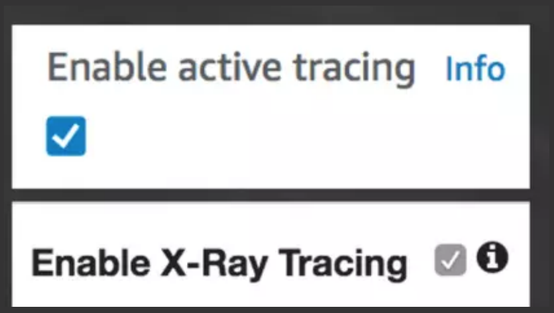
Zappa

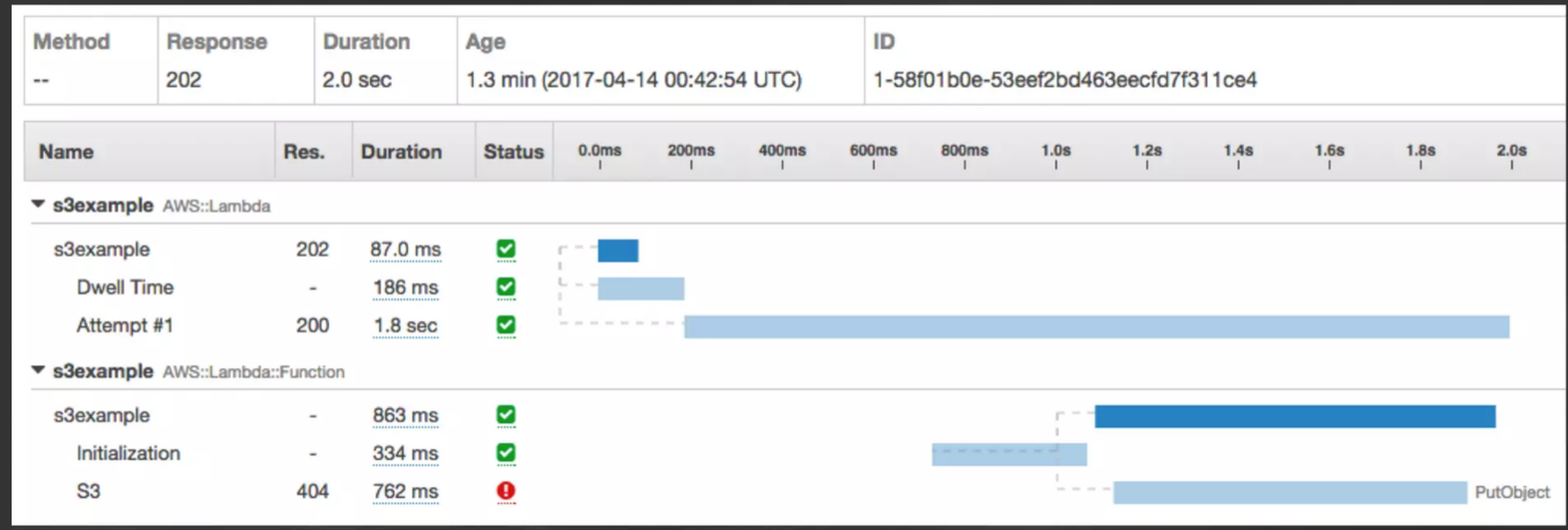
Metrics and Logging in AWS Lambda

- AWS CloudWatch Integration: Lambda automatically integrates with CloudWatch for logging and monitoring.
- Real-Time Metrics: Track function invocations, errors, duration, and concurrent executions.
- Detailed Logging: Gain insights into execution with CloudWatch Logs, including request, response, and error data.
- Custom Metrics and Alarms: Create custom metrics and set alarms for proactive monitoring and response.

AWS X-Ray Integration with Serverless

- Functionality: Provides insights into the behavior of your serverless applications.
- Performance Analysis: Traces and analyzes requests as they travel through your AWS services.
- Error Identification: Helps identify and troubleshoot performance bottlenecks and issues.
- Service Integration: Seamlessly integrates with AWS Lambda, Elastic Beanstalk, and other AWS services.





Challenges and Solutions in AWS Lambda

Cold Starts: Issue: Initial latency during the first function execution.	Solution: Use provisioned concurrency, optimize startup code, schedule regular invocations to keep functions warm.
Debugging Complexities: Issue: Difficulties in troubleshooting ephemeral Lambda functions.	Solution: Employ AWS CloudWatch Logs, AWS X-Ray for tracing, structured logging.
Execution Limits: Issue: Limits on execution time, memory, and package size.	Solution: Modularize functions, optimize settings, use AWS Step Functions for orchestration.