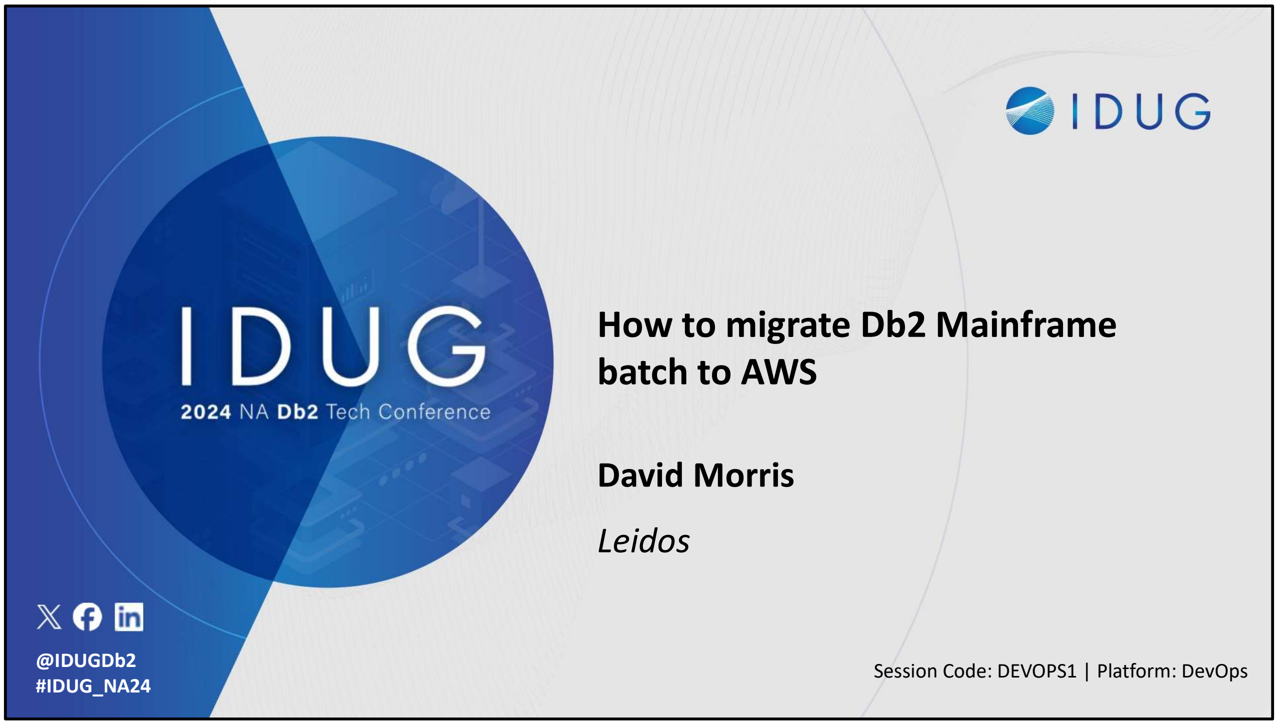


IDUG

IDUG

2024 NA **Db2** Tech Conference

How to migrate Db2 Mainframe batch to AWS

David Morris

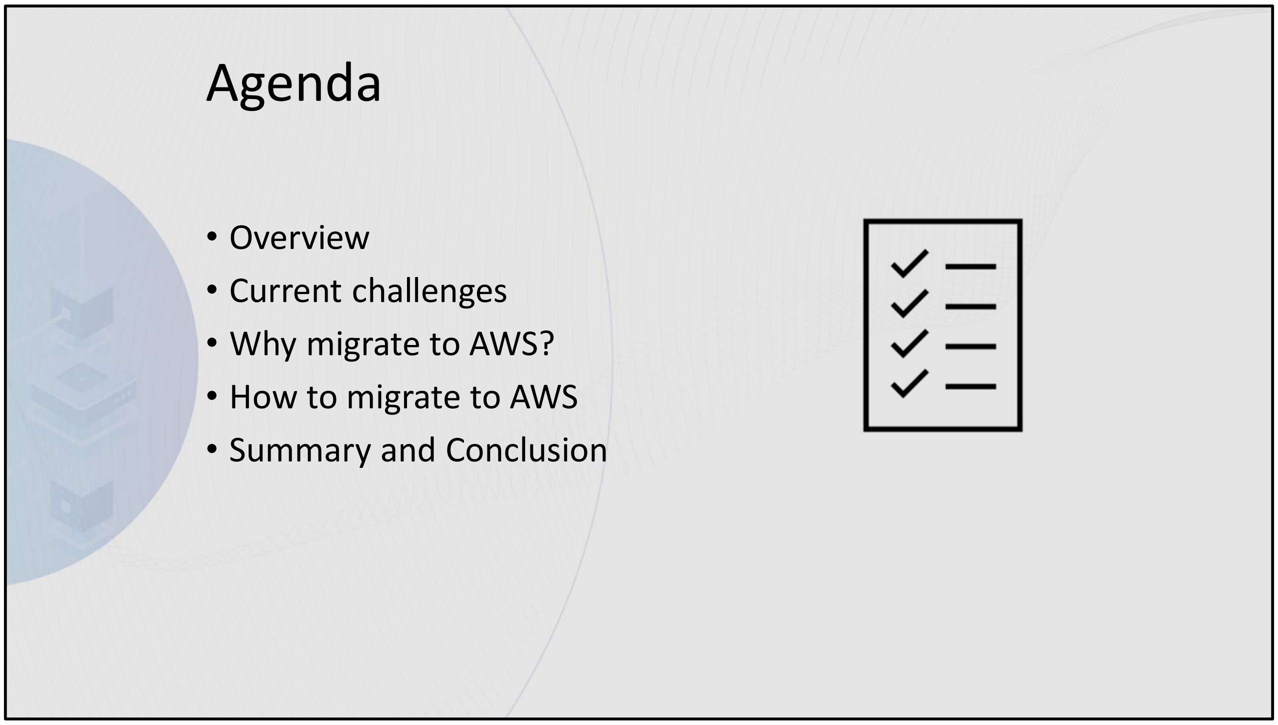
Leidos

@IDUGDb2
#IDUG_NA24

Session Code: DEVOPS1 | Platform: DevOps

My name is David Morris I been a Leidos employee for over 15 years and over 30 years of IT development experience. I work on a large U.S. government contract using applications that access Db2 databases. Today's database presentation is 'how to migrate Db2 Mainframe batch to AWS'. I have been on projects where mainframe legacy applications have been migrated successfully to modern environments and modern development include AWS gov cloud.

Previous AWS presentations have included Db2 in AWS cloud. Today's presentation will be an on-prem Db2 database with batch migration. Reflecting on my previous IDUG talk on Spring Batch, it's evident that technology continually evolves; currently, AWS Batch technology is at the forefront.



Agenda

- Overview
- Current challenges
- Why migrate to AWS?
- How to migrate to AWS
- Summary and Conclusion

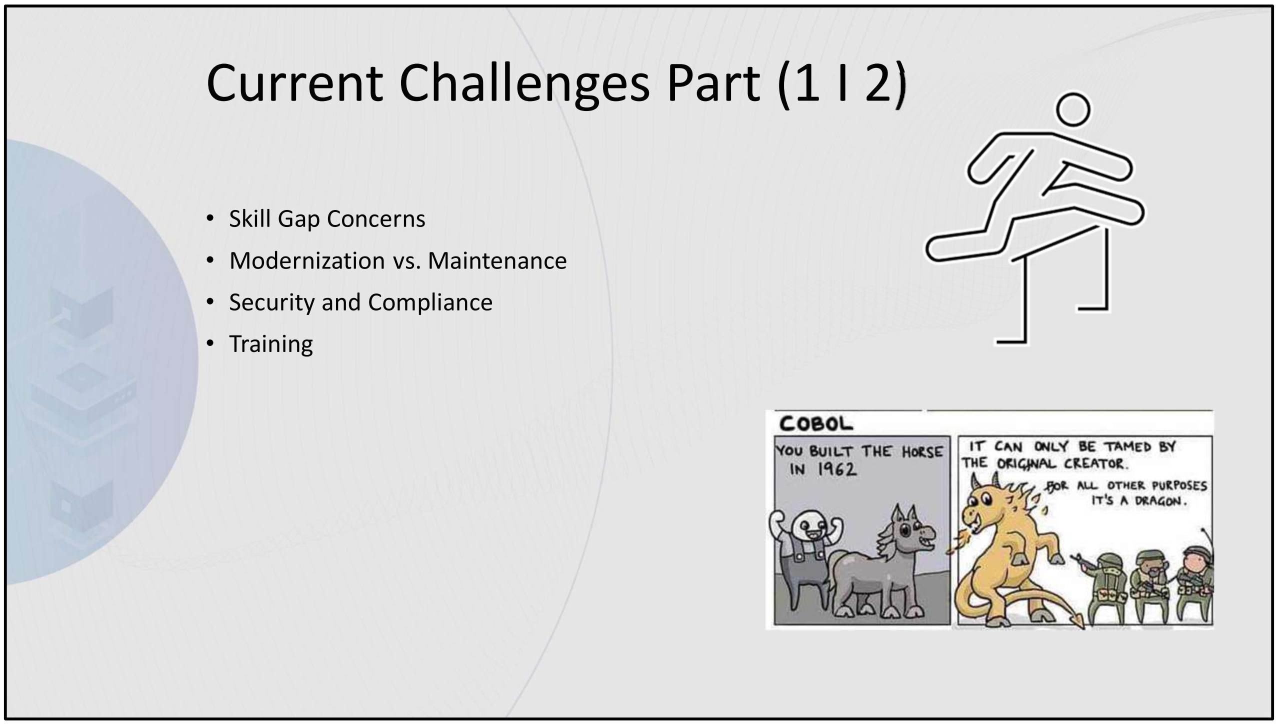
Overview – Today's discussion will focus on the architectural design of a Db2 database that utilizes AWS services for executing Db2 batch processes, transitioning from a legacy on-premises system to a cloud-based solution. We will explore the potential risks associated with this migration. The session will specifically address the migration of Db2 Mainframe batch operations from an on-premises environment to AWS, considering a scenario where the Db2 database connects to AWS web services through a Direct Connection over the AWS network.

Current challenges – current challenges faced by AWS migration. On-prem to cloud.

Why migrate to AWS? Why do we want to migrate? Is it just to modernize? What are the benefits?

How to migrate to AWS. What are best practices? What are the risks involved.?

Followed up by a Summary and Conclusion. Please save any questions you may have at the end of the presentation. We can revisit any slides.



Current Challenges Part (1 I 2)

- Skill Gap Concerns
- Modernization vs. Maintenance
- Security and Compliance
- Training

COBOL
YOU BUILT THE HORSE IN 1962
IT CAN ONLY BE TAMED BY THE ORIGINAL CREATOR.
FOR ALL OTHER PURPOSES IT'S A DRAGON.

COBOL (Common Business-Oriented Language) has been a stalwart in the world of business data processing for over six decades. Despite the emergence of newer technologies, COBOL continues to play a critical role ,especially in finance and government sectors. Let's delve into some of the challenges faced by on-premises COBOL batch systems:

1. **Skill Gap Concerns**:
   - One of the most pressing issues is the diminishing pool of knowledgeable COBOL programmers. As the workforce ages out
   - Organizations struggle to find skilled COBOL developers who can maintain, update, and secure existing COBOL systems

One of the government employees asked me if I knew of any COBOL developers. I said sure I know a few. Great she said. Can I have their contact details, to which I responded "they are all part of our current workforce".

2. **Modernization vs. Maintenance**:
   - The debate between modernizing COBOL systems and maintaining them in their current state is complex.
   - Modernization can be costly and risky, especially for systems handling critical operations. On the other hand, maintenance requires skilled personnel and can be challenging as technology advances[1].

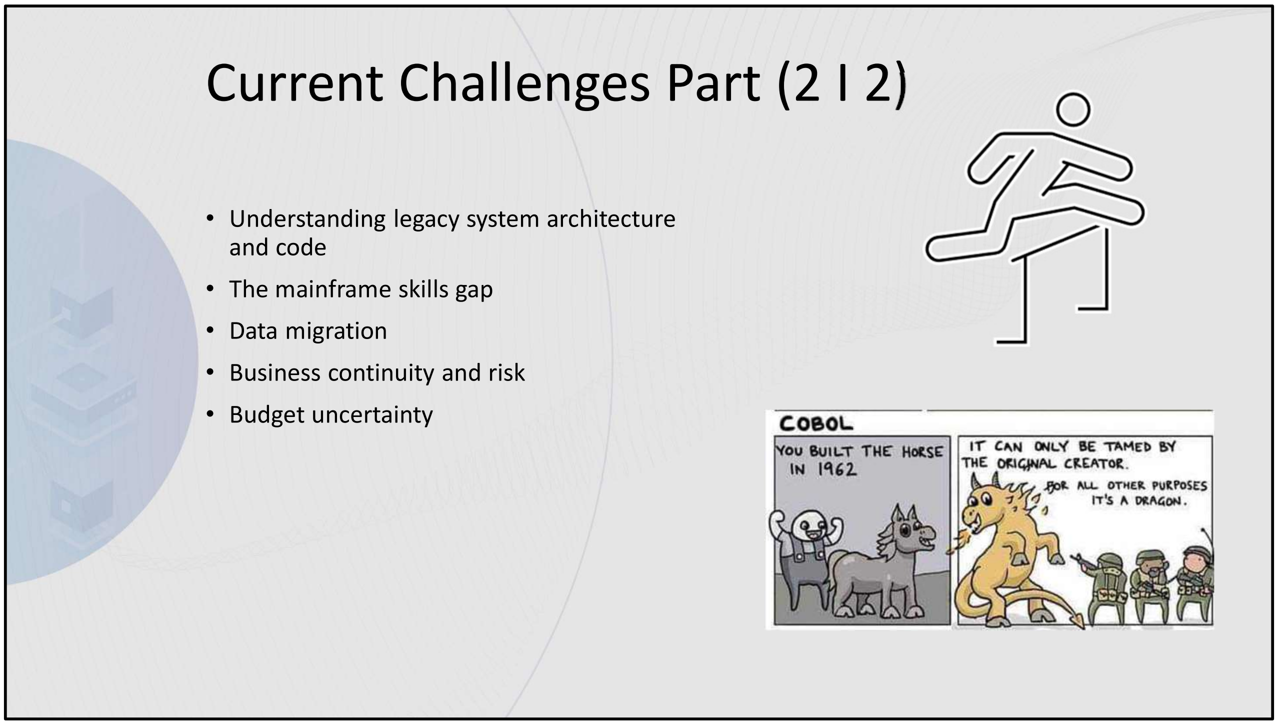
Point to Horse cartoon – "You built the horse in 1962 by original developer. For all others it has morphed into a dragon".

3. **Security and Compliance**:
   - Ensuring that COBOL systems remain secure and compliant with current data protection and privacy standards is an ongoing challenge.
   - As threats evolve, maintaining the integrity and security of these systems becomes paramount[1].

Currently in the CIO's office for Enterprise architecture worked on MFA. ZMFA for PCOMM screens.

5. **Renewed Interest in Training**:
   - Recognizing the skill gap, corporate and governments have emphasized the need for COBOL programmers. Not teaching COBOL in University. The new technologies are Python, JavaScript, Web and cloud. Sure, there is Java and .NET. Java was released in 1995.

In summary, while COBOL remains essential, addressing the skill gap, balancing modernization efforts, ensuring security are needed for migration of COBOL batch systems to more modern processes.



Current Challenges Part (2 I 2)

- Understanding legacy system architecture and code
- The mainframe skills gap
- Data migration
- Business continuity and risk
- Budget uncertainty

COBOL
YOU BUILT THE HORSE IN 1962
IT CAN ONLY BE TAMED BY THE ORIGINAL CREATOR.
FOR ALL OTHER PURPOSES IT'S A DRAGON.

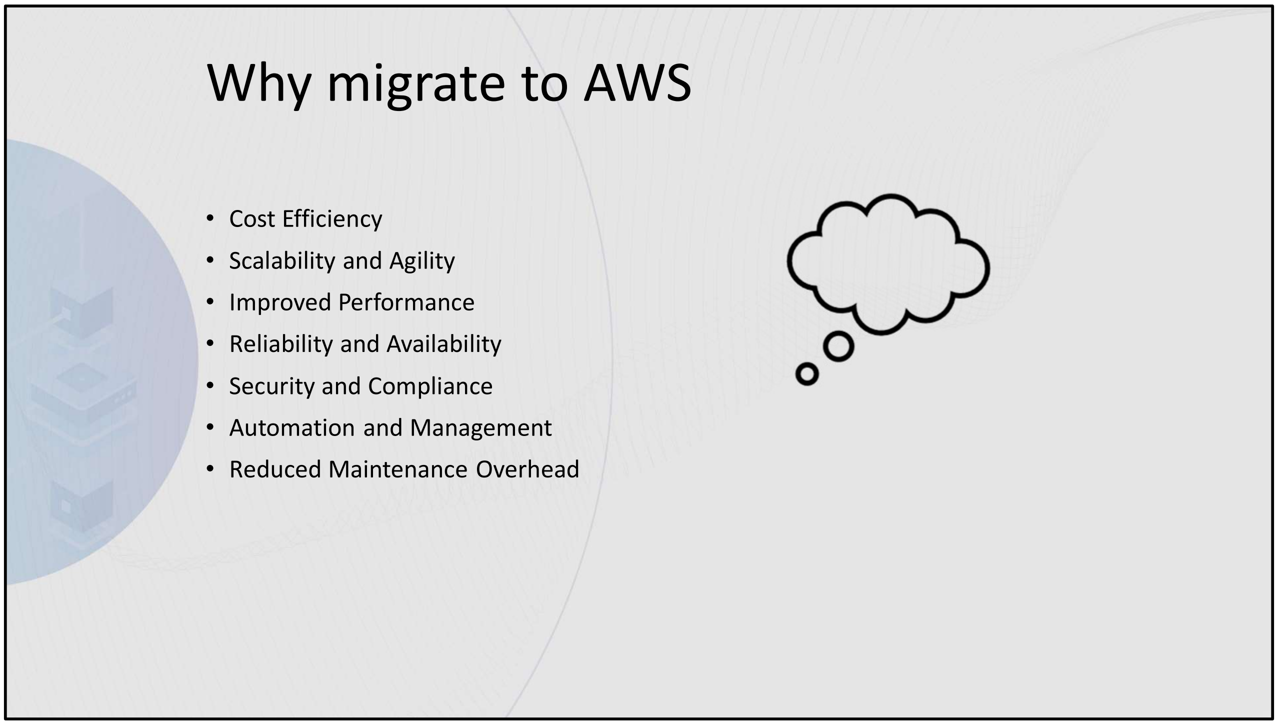
Legacy Systems: Understanding the architecture and code of legacy systems is crucial for modernization efforts. It involves analyzing outdated technologies and identifying risks and inefficiencies.

Mainframe Skills Gap: Addressing the shortage of skilled mainframe professionals is essential as many experts retire. Strategies include education, training, and adopting new technologies like cloud.

Data Migration: Moving data between different storage systems or computing environments is a key IT management task, especially when adopting new technologies or consolidating data centers

Business Continuity: Developing a Business Continuity Plan (BCP) helps organizations prepare for and recover from potential threats, ensuring operational integrity and resilience

Budget Uncertainty: Managing uncertainty in budgeting is important for measurement quality and decision-making. It involves creating an uncertainty budget to identify and quantify sources of uncertainty.



Why migrate to AWS

- Cost Efficiency
- Scalability and Agility
- Improved Performance
- Reliability and Availability
- Security and Compliance
- Automation and Management
- Reduced Maintenance Overhead

Cost Efficiency: AWS provides a pay-as-you-go model, allowing businesses to scale resources up or down based on demand. By migrating to AWS, you can avoid the upfront costs associated with maintaining on-premises infrastructure and only pay for the resources you actually use.

Scalability and Agility: AWS offers elastic compute resources, enabling you to easily scale your batch processing capabilities. Whether you need to process large amounts of data during peak times or handle smaller workloads, AWS can accommodate your needs.

Improved Performance: AWS provides high-performance computing options, such as EC2 instances optimized for specific workloads. You can choose the right instance type to meet your batch processing requirements, resulting in faster execution times.

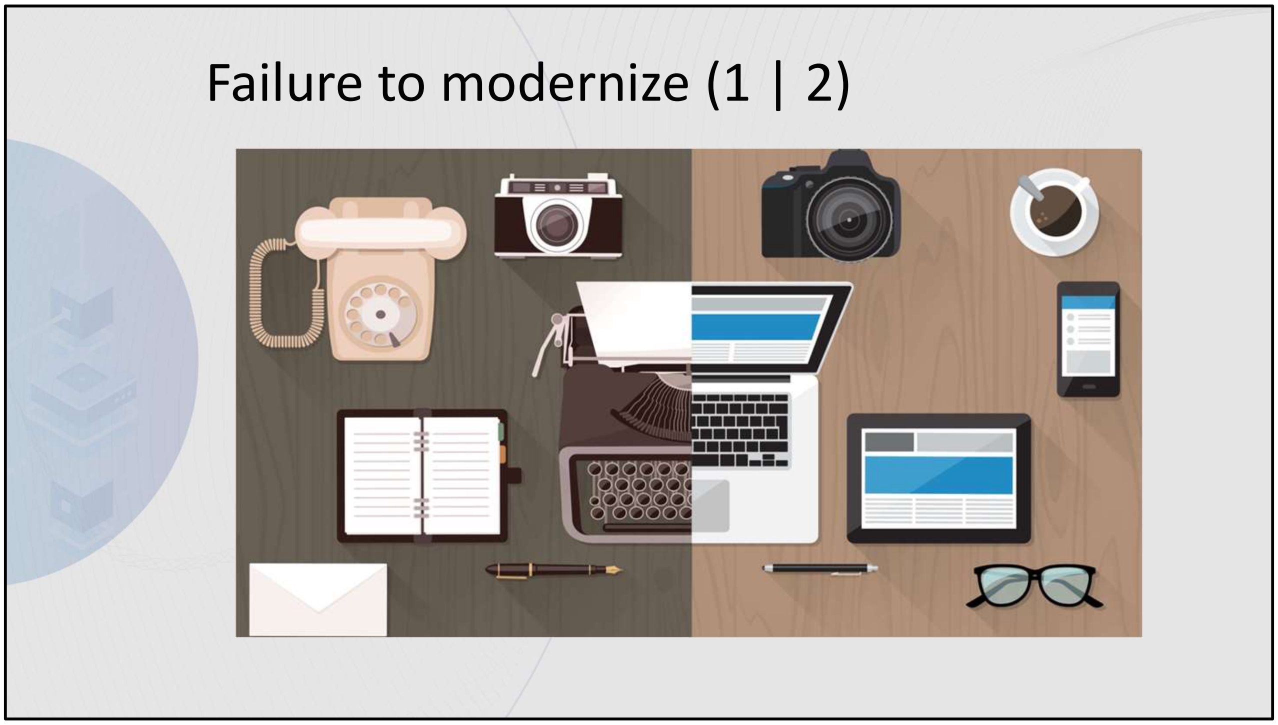
Reliability and Availability: AWS offers redundancy across multiple availability zones, ensuring high availability for your batch processes. Additionally, services like Amazon S3 provide durable storage for your data.

Security and Compliance: AWS provides robust security features, including encryption, access controls, and compliance certifications. By migrating to AWS, you can enhance the security of your batch processes.

Automation and Management: AWS services like AWS Batch and AWS Step Functions allow you to automate batch processing workflows. You can define complex job dependencies, manage retries, and monitor job progress.

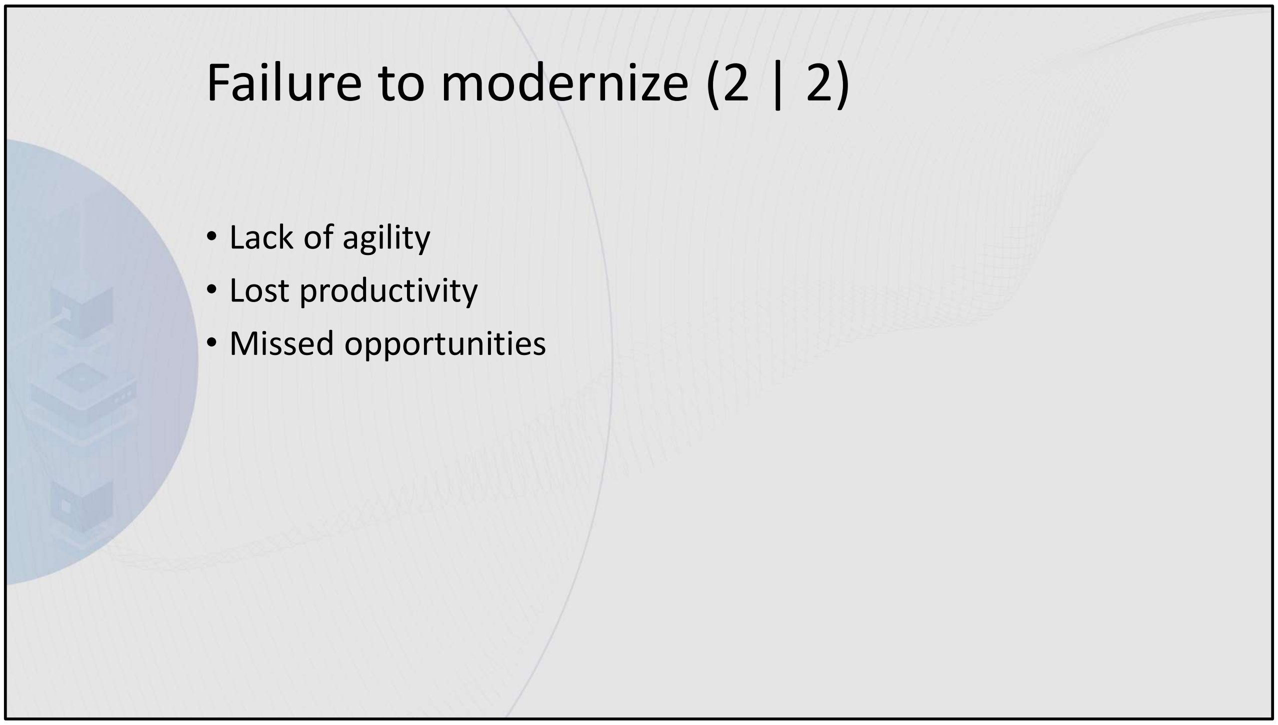
Reduced Maintenance Overhead: With AWS, you no longer need to manage physical servers, perform hardware upgrades, or handle routine maintenance tasks. AWS takes care of infrastructure management, allowing your team to focus on business logic and application development.

Remember that each migration is unique, and it's essential to assess your specific business goals, workload dependencies, and regulatory requirements before planning the migration. Collaborate with stakeholders and follow a step-by-step process to ensure a successful transition.



Failure to modernize (1 | 2)

Compare pictures on left to pictures on right



Failure to modernize (2 | 2)

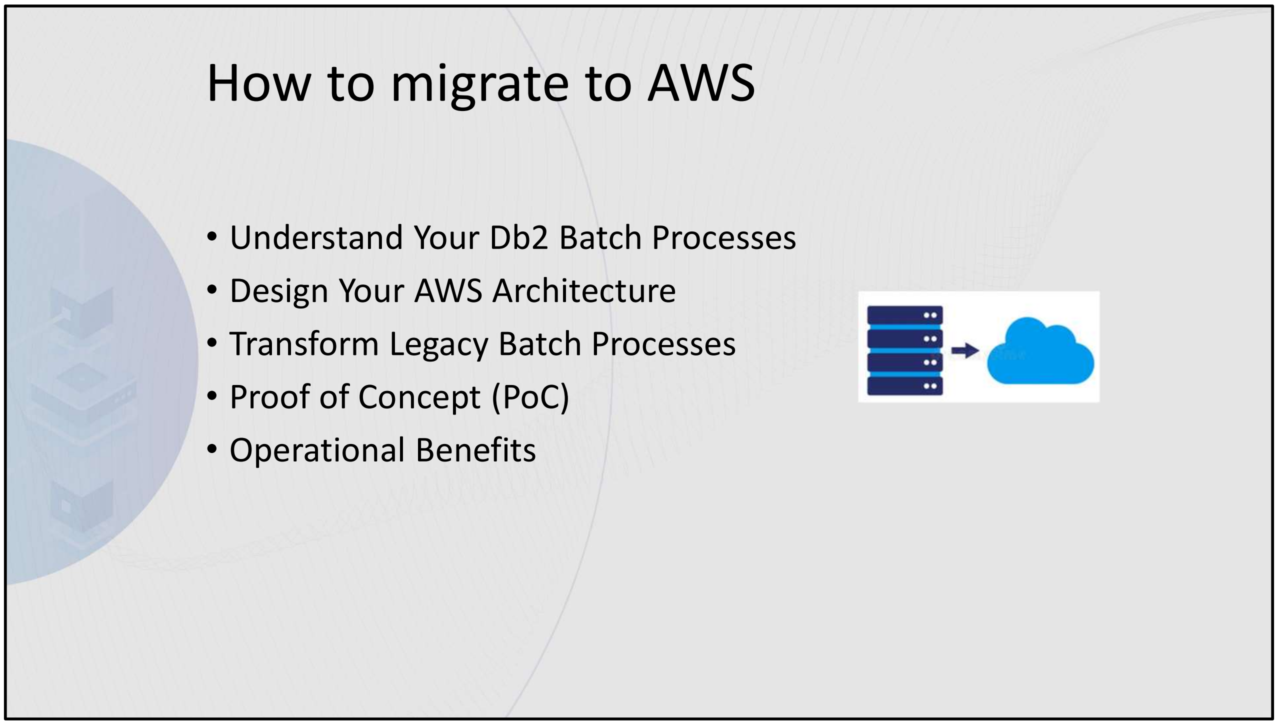
- Lack of agility
- Lost productivity
- Missed opportunities

The key risks organizations face when they fail to modernize critical business applications are:

Lack of agility: Relying on legacy technology can mean lacking the flexibility to tackle key business challenges or lagging behind other organizations facing similar hurdles.

Lost productivity: By failing to integrate with leading applications or leading to workarounds, relying on outdated technology often results in unnecessarily complex, inefficient business processes.

Missed opportunities: Sticking with legacy applications means your business won't be able to deliver new ways of delighting customers or enable new ways of working. Failing to take advantage of leading-edge capabilities puts you behind the competition, making you less able to compete for both customers and talent.



How to migrate to AWS

- Understand Your Db2 Batch Processes
- Design Your AWS Architecture
- Transform Legacy Batch Processes
- Proof of Concept (PoC)
- Operational Benefits

Understand your Db2 Batch processes
- Analyze your existing mainframe batch processes. Identify the critical components, data dependencies, and performance requirements.
- Consider the specific technologies used in your mainframe environment, such as z/OS CICS, JCL, Cobol, DB2, VSAM files

Design your AWS Architecture
- Microservices offer agility, scalability, and innovation. However, they also introduce operational complexity. Whether to use Lamda for short running processes to EC2 container with source code

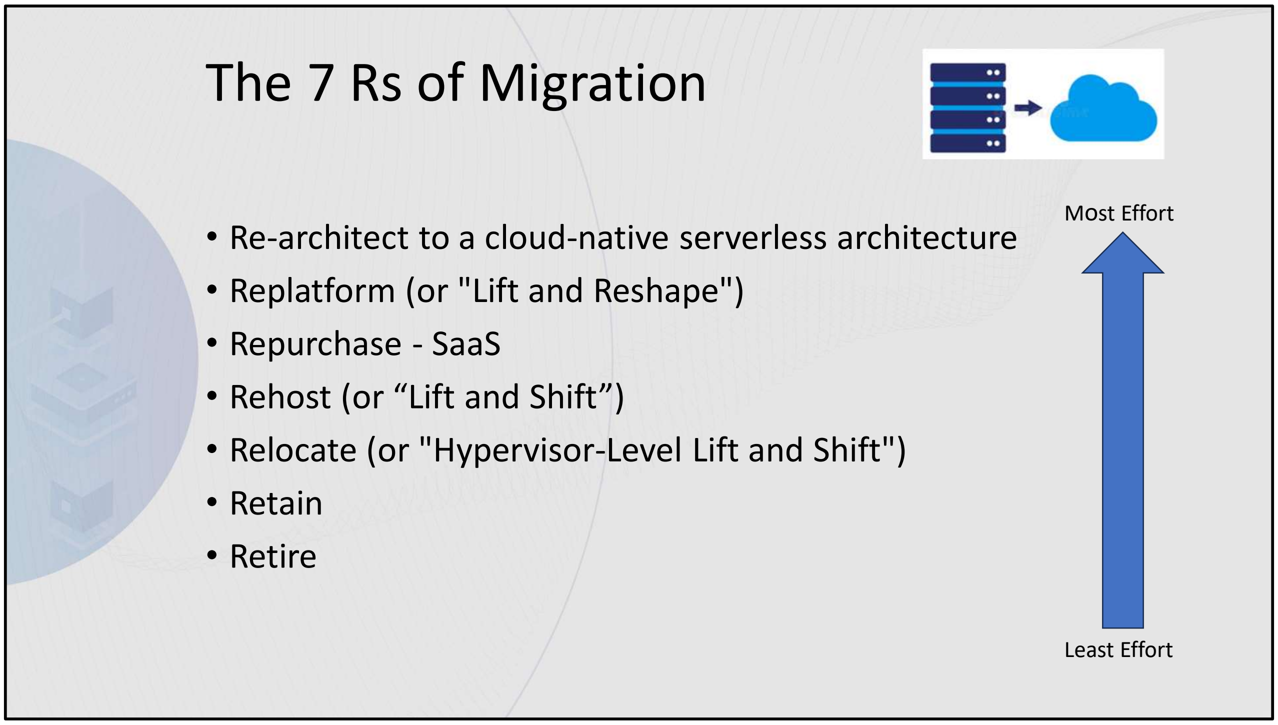
Transform Legacy Batch Processes
- Convert your mainframe batch processes into real-time microservices.
- Leverage AWS services to achieve this transformation

Developing a **Proof of Concept (PoC)** is crucial in software development for several reasons:
- **Feasibility Assessment**: A PoC validates your project idea early on. It helps determine if the concept can be built within budget and technical constraints.

- **Stakeholder Trust**: stakeholders need evidence before committing resources. A PoC builds trust by demonstrating viability
- **Better Planning**: Thinking ahead during the PoC phase allows teams to solve logistical issues before full-scale development

Operational Benefits

- By migrating to AWS, you can achieve a more efficient and cost-optimized architecture.
- AWS provides on-demand resources, infrastructure as code, and continuous delivery.
- The transformation allows you to move from monolithic mainframe batch processing to real-time microservices.



The 7 Rs of Migration

- Re-architect to a cloud-native serverless architecture
- Replatform (or "Lift and Reshape")
- Repurchase - SaaS
- Rehost (or “Lift and Shift”)
- Relocate (or "Hypervisor-Level Lift and Shift")
- Retain
- Retire

Most Effort
Least Effort

7 Rs of cloud migration are a set of strategies designed to help organizations plan, execute, and optimize their migration projects. These strategies provide a roadmap for determining the best approach to moving applications and data from on-premises infrastructure to the cloud.

Starting from the bottom up. Least effort to Most effort.

Retire:
- Evaluate existing applications and data to determine if they are still necessary.
- Retire any workloads that are no longer relevant or valuable.
- Decommissioning unused systems reduces complexity and costs.

Retain:
- Not all applications and data need to move to the cloud.
- Some workloads may be better suited to remain on-premises due to regulatory requirements, performance constraints, or other factors.
- The retain strategy allows organizations to selectively keep certain workloads in their current environment.

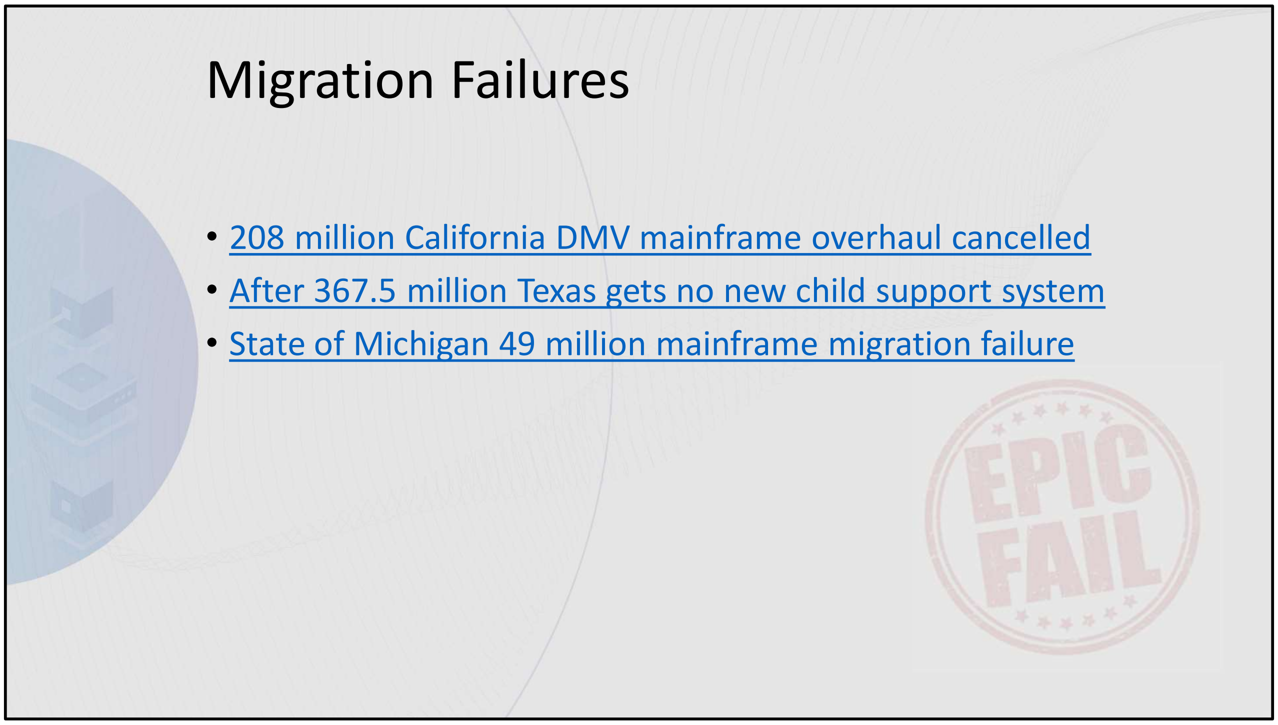
Relocate:
   - Similar to rehosting, relocation involves moving VMs to the cloud.
   - However, it allows for minor adjustments, such as resizing VMs or changing the underlying hypervisor.
   - This strategy aims to optimize performance and resource utilization during migration.

Rehost:
   - In this strategy, you move your existing applications and data to the cloud without making significant changes.
   - It involves migrating virtual machines (VMs) as-is, maintaining the same architecture and functionality.
   - Rehosting is often a quick way to get workloads into the cloud but may not fully leverage cloud-native features[1].

Repurchase:
   - In this strategy, you replace existing software with a cloud-based alternative.
   - For example, migrating from an on-premises CRM system to a Software-as-a-Service (SaaS) CRM solution.
   - Repurchasing can simplify management and reduce maintenance overhead.

Replatform:
   - Replatforming involves making some modifications to your applications during migration.
   - You adapt the application to take advantage of cloud services while minimizing code changes.
   - For example, you might move a database from an on-premises server to a managed database service in the cloud.

Re-architect:
   - Refactoring is a more significant transformation.
   - It involves redesigning parts of your application to fully leverage cloud-native capabilities.
   - You might break monolithic applications into microservices, use serverless functions, or adopt containerization.



Migration Failures

- 208 million California DMV mainframe overhaul cancelled
- After 367.5 million Texas gets no new child support system
- State of Michigan 49 million mainframe migration failure

The $208 million California DMV mainframe overhaul project was cancelled due to several issues:

- **Lack of Progress**: Significant concerns arose regarding the project's advancement
- **Contractual Issues**: Hewlett-Packard's contract ended, and they refused to hire key staff until renegotiation
- **Historical Challenges**: The DMV had previously experienced a failed IT project, indicating a pattern of difficulties in modernization efforts

These factors contributed to the decision to halt the project

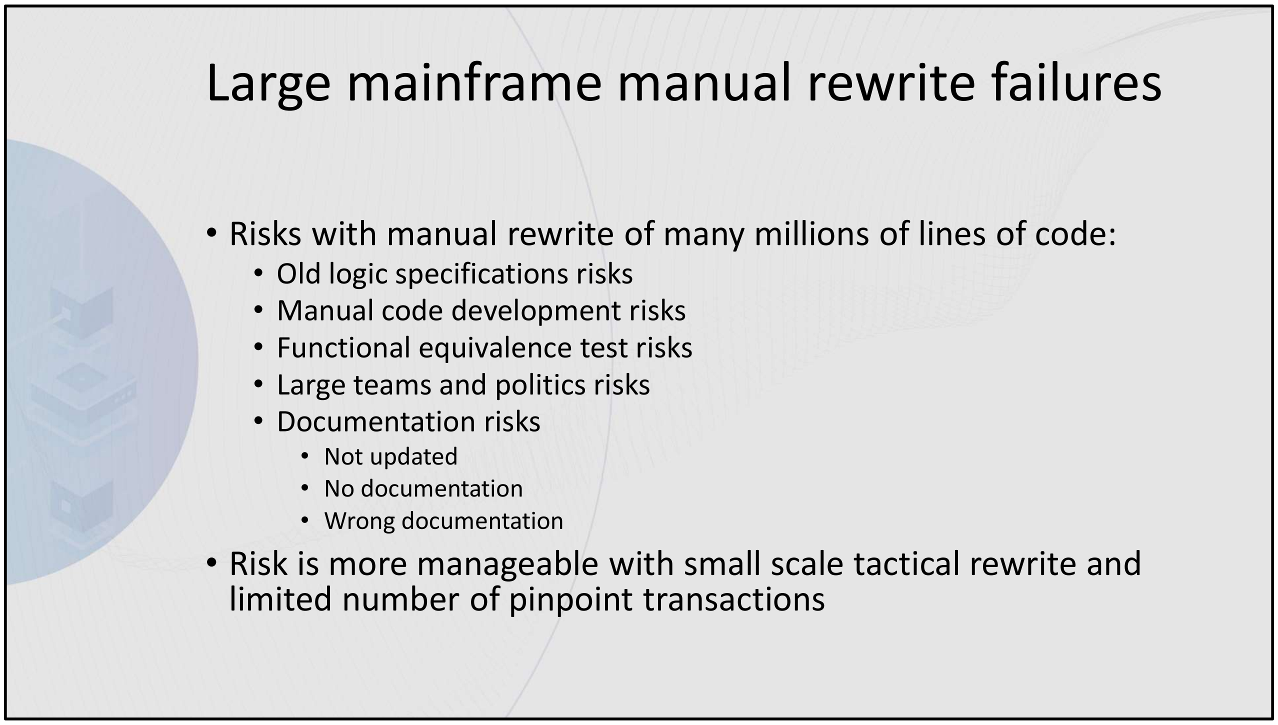
The failure of the Texas child support system overhaul, despite the investment of $367.5 million, can be attributed to several factors:

- **Poor Design**: The system's design was not up to the required standards.
- **Insufficient Skills**: There was a lack of sufficiently skilled labor to execute the project.
- **Budget Overrun**: The project's budget ballooned from the initial estimate, leading to financial concerns.

These issues highlight the complexities and challenges associated with large-scale IT projects

The State of Michigan’s $49 million mainframe migration project faced failure due to several reasons:

- **Contractual Disputes**: The state sued Hewlett-Packard (HP) for failing to meet contract requirements
- **Project Delays**: The project experienced multiple deadline extensions and remained unfinished
- **Termination of Contract**: Michigan terminated the contract after fruitless negotiations, claiming HP failed to deliver any new functions despite payments



Large mainframe manual rewrite failures

- Risks with manual rewrite of many millions of lines of code:
  - Old logic specifications risks
  - Manual code development risks
  - Functional equivalence test risks
  - Large teams and politics risks
  - Documentation risks
    - Not updated
    - No documentation
    - Wrong documentation
- Risk is more manageable with small scale tactical rewrite and limited number of pinpoint transactions

Manual rewrites of large mainframe applications have historically faced numerous challenges and often resulted in failures. Here's a summary based on research:

Old logic specifications risks: May lack clarity or be incomplete due to the passage of time, changes in personnel, or poor documentation practices. Lost of tribal knowledge when someone leaves or retires.

Manual code development risks: Error-prone. Manual code rewrite involves a high risk of errors due to human oversight or mistakes. Since mainframe code can be complex and intricate, even minor errors can lead to significant issues in the software.

Functional equivalence test risks: the challenges and pitfalls associated with ensuring that the rewritten code performs identically to the original code in terms of functionality. Mainframe systems often consist of highly complex and interconnected components, making it challenging to accurately replicate the behavior of the original code during the rewrite process. This complexity introduces a significant risk of overlooking certain functionalities or edge cases.

Large teams and politics risks: the involvement of large teams and organizational politics can introduce several risks that may impact project success.

- Communication
- Coordination
- Resistance to Change
- Resource Allocation
- Lack of Accountability

1.**Complexity Overwhelm**: Mainframe applications tend to be highly complex, with extensive codebases and intricate dependencies. Manual rewriting efforts often struggle to comprehensively understand and replicate this complexity, leading to incomplete or inaccurate translations.
2.**Resource Intensiveness**: Manual rewriting requires significant resources in terms of time, manpower, and expertise. Teams may underestimate the effort required, leading to delays, cost overruns, and ultimately project abandonment.
3.**Functional Equivalence Issues**: Ensuring that the rewritten application behaves identically to the original is challenging. Variations in interpretation, overlooked functionalities, or differences in business logic can lead to functional discrepancies that impact operations.
4.**Lack of Maintenance**: Mainframe applications typically lack adequate documentation and have undergone years of ad-hoc modifications, leading to "spaghetti code" and accumulated technical debt. Manual rewrites may fail to address this underlying maintenance issue, perpetuating problems in the new system.
5.**Testing Complexity**: Verifying the correctness and reliability of a manually rewritten application is daunting. Comprehensive testing is essential but can be time-consuming and costly, particularly for large and complex systems.
6.**Loss of Domain Knowledge**: Mainframe applications often represent decades of institutional knowledge embedded in the codebase. Manual rewriting risks losing this domain expertise if not adequately captured and transferred during the process.
7.**Disruption to Operations**: Attempting a manual rewrite can disrupt ongoing operations, leading to downtime, user dissatisfaction, and potential business disruptions. The longer the manual rewrite takes, the more disruptive it becomes.
8.**Technological Obsolescence**: Manual rewrites may adopt outdated technologies or design patterns, missing opportunities for modernization and

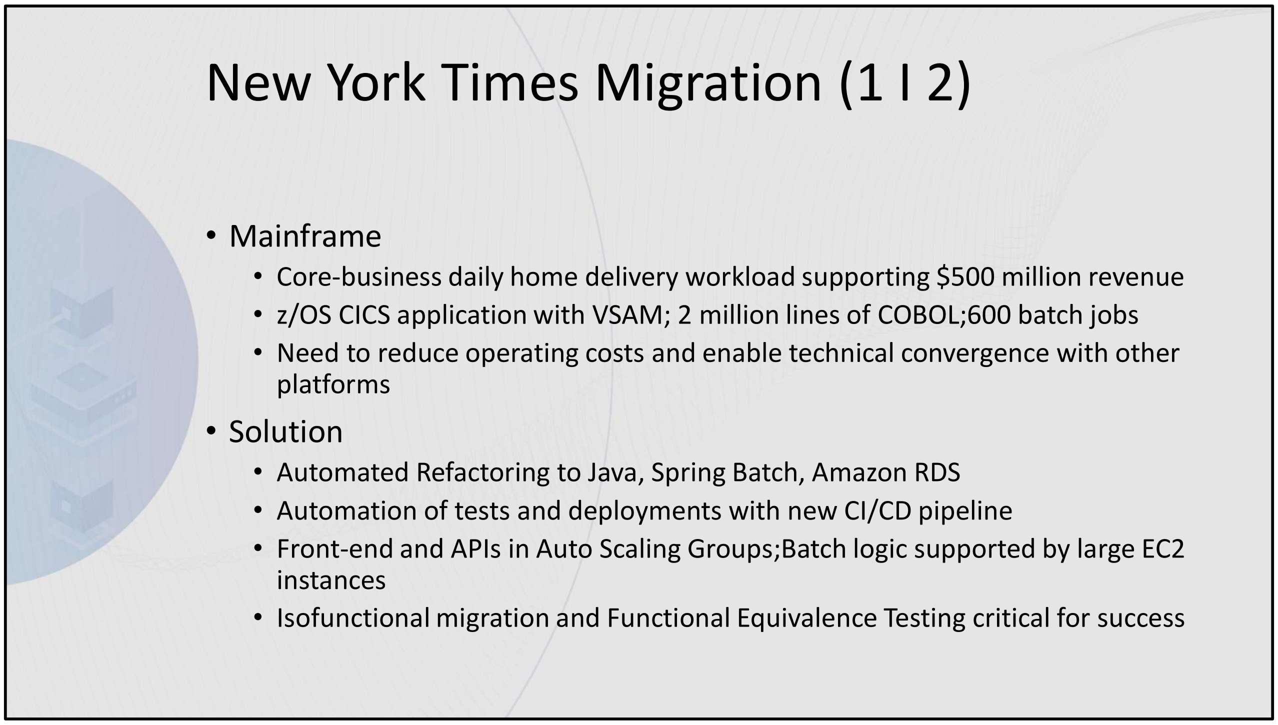
optimization available in newer platforms or architectures.
In summary, manual rewrites of large mainframe applications often fail due to their complexity, resource intensiveness, challenges in achieving functional equivalence, lack of maintenance, testing complexities, loss of domain knowledge, operational disruptions, and risks of technological obsolescence. As a result, alternative approaches such as automated refactoring or gradual modernization strategies are increasingly favored for mitigating these risks and achieving successful transformations.

Rip and Replace is Risky

The “Rip and Replace” approach for migrating from mainframe to AWS is considered risky due to several factors:

- **Complexity**: Mainframes often run intricate, mission-critical applications developed over many years, making migration without business disruption challenging
- **Compatibility Issues**: Potential compatibility problems between old mainframe applications and the new cloud environment can lead to unexpected migration issues
- **Data Loss**: The risk of data loss when transferring large volumes of data, especially if data structures differ significantly between systems
- **Downtime**: Migration might require system downtime, affecting business continuity
- **Security Concerns**: New security vulnerabilities could emerge during the transition if not properly managed

For a successful migration, careful planning and execution are essential to mitigate these risks.



New York Times Migration (1 I 2)

- Mainframe
  - Core-business daily home delivery workload supporting $500 million revenue
  - z/OS CICS application with VSAM; 2 million lines of COBOL;600 batch jobs
  - Need to reduce operating costs and enable technical convergence with other platforms
- Solution
  - Automated Refactoring to Java, Spring Batch, Amazon RDS
  - Automation of tests and deployments with new CI/CD pipeline
  - Front-end and APIs in Auto Scaling Groups;Batch logic supported by large EC2 instances
  - Isofunctional migration and Functional Equivalence Testing critical for success

The core IT system supporting The New York Times' daily Home Delivery Platform, running on a mainframe, faced critical business demands. To modernize and optimize operations, the legacy COBOL-based application underwent a successful transformation into a Java-based system hosted on Amazon Web Services (AWS). Through innovative automated refactoring, the application transitioned to object-oriented code, and its data was migrated from legacy indexed-files to a relational database.

The migration from the IBM Z mainframe, with its z/OS operating system, was driven by the need to reduce operating expenses and align the Digital Platform with the Home Delivery Platform. Previous attempts to manually rewrite the application proved unsuccessful, leading to the adoption of a strategy focused on automated refactoring for code and data migration. This approach ensured functional equivalence, cost reduction, and seamless integration with modern technologies.

The mainframe application handled crucial functionalities such as billing, invoicing, customer accounts, delivery routing, product catalog management, pricing, and financial reporting. It operated as a CICS/COBOL application with a

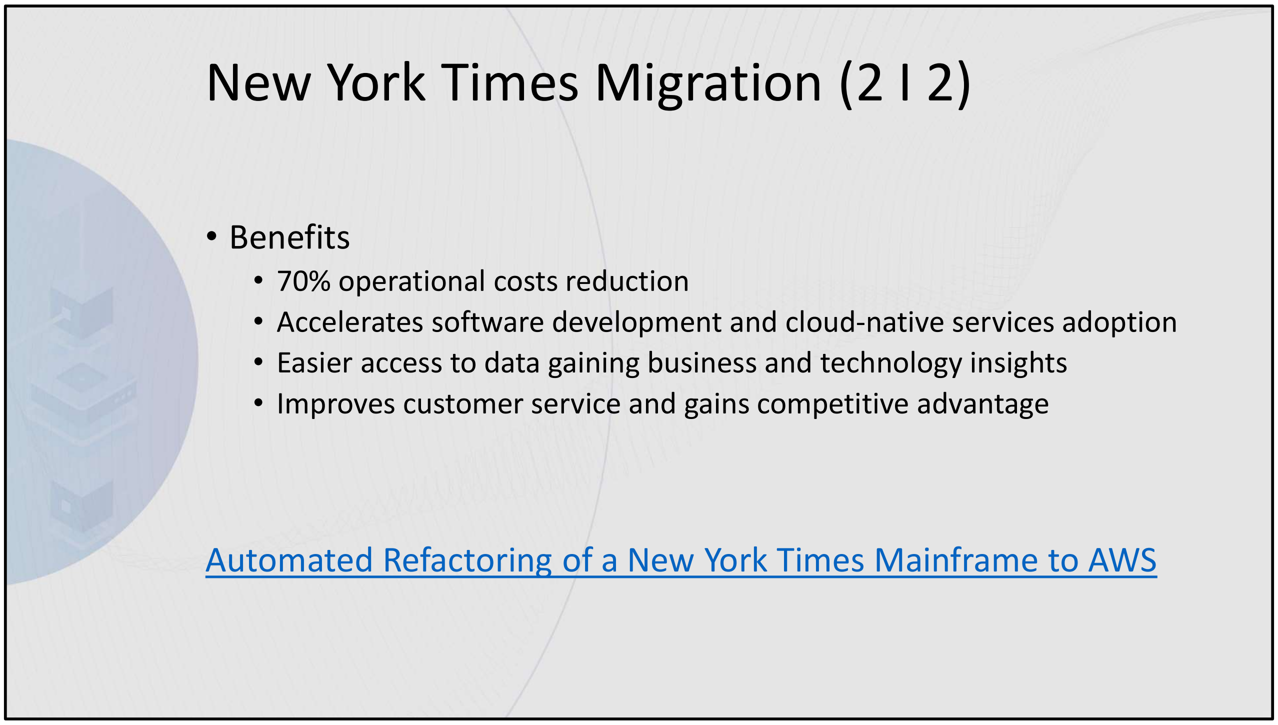
BMS-based 3270 interface accessing VSAM KSDS business data, supported by batch processing via JCL jobs with CA7 for job scheduling.
With over two million lines of COBOL code, 600 batch jobs, and 3,500 files transmitted daily to downstream systems, the legacy system consumed substantial storage and resources. Transitioning to Java involved converting each COBOL program into a Java class, with JCL transformed into JSR-352 XML using the Spring Batch runtime. VSAM KSDS files were migrated to an Oracle relational database, streamlining data management.

Testing, constituting a significant portion of the project timeline, was crucial for ensuring functional equivalence between the Java and COBOL applications. High test coverage was imperative, necessitating the automation of test case creation and analysis, particularly for batch jobs.
The migration to the cloud, specifically AWS, became the preferred deployment environment for The New York Times, marking a strategic shift. Aristo, the transformed application, transitioned from a private data center to AWS within a year of operation.

To expedite releases, a Continuous Integration and Continuous Delivery (CI/CD) pipeline was established. This phase encompassed the COBOL-to-Java transformation and the VSAM-to-relational database conversion, culminating in the launch of the Aristo application in production on-premises.

Future improvements for The New York Times include breaking down the application monolith into microservices. Key lessons learned emphasized the importance of comprehensive testing, identification, and removal of obsolete code, cross-training of developers, and meticulous application understanding during analysis and planning phases.

Overall, the modernization project evolved from a cost-cutting initiative into a strategic move to leverage advanced technology for improved customer service and competitive advantage in the dynamic media industry.

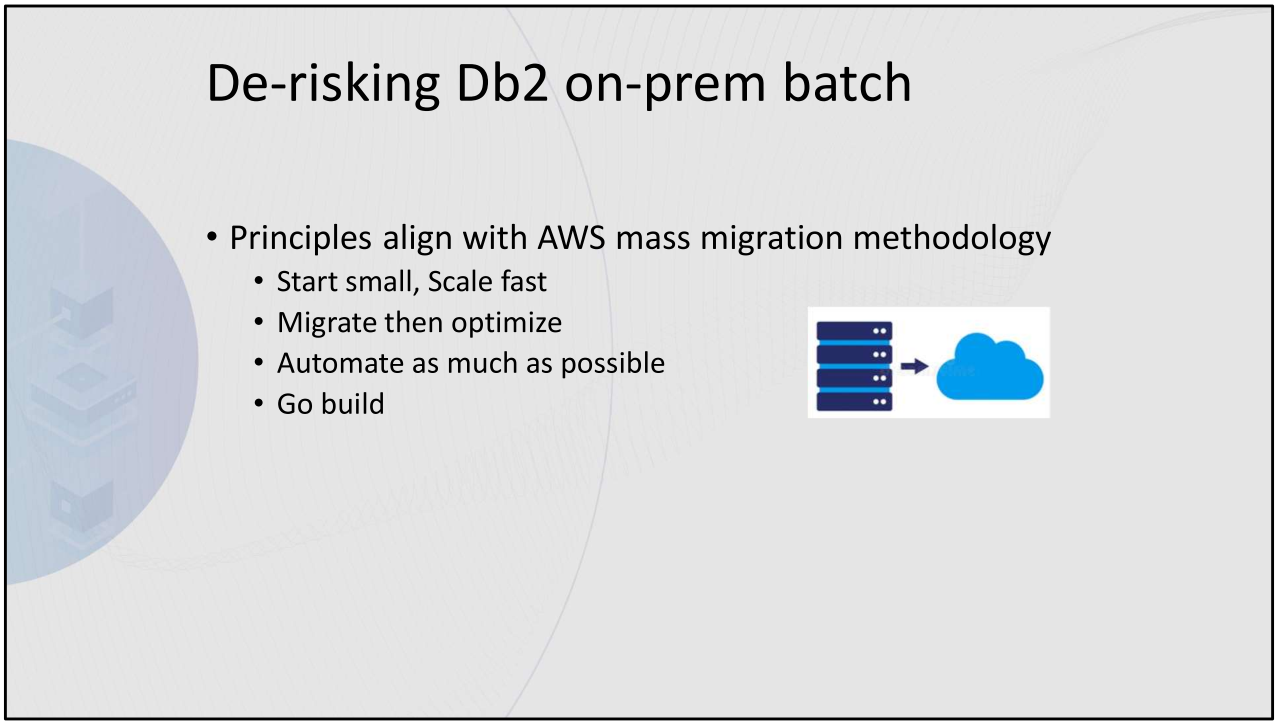


New York Times Migration (2 I 2)

- Benefits
  - 70% operational costs reduction
  - Accelerates software development and cloud-native services adoption
  - Easier access to data gaining business and technology insights
  - Improves customer service and gains competitive advantage

Automated Refactoring of a New York Times Mainframe to AWS

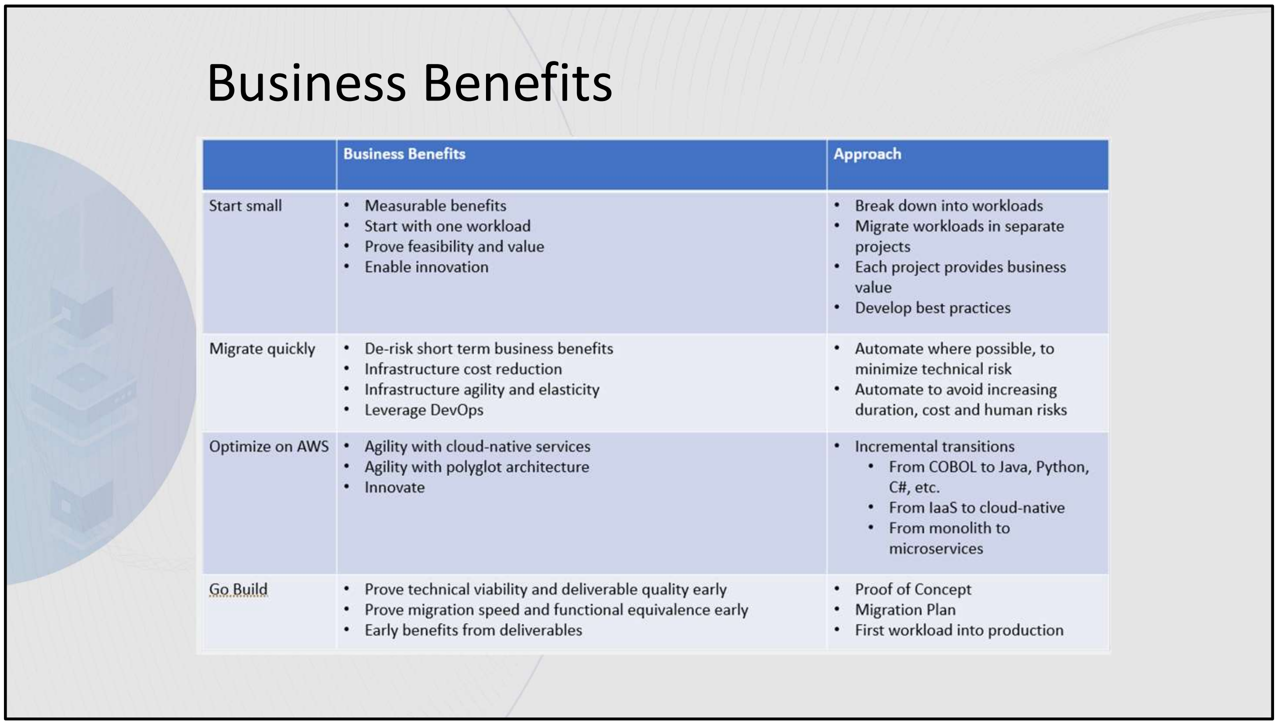
The modernization of The New York Times' core IT system supporting its Home Delivery Platform resulted in significant benefits. The legacy COBOL-based application was successfully transformed into a Java-based system hosted on AWS, reducing operating expenses and aligning with modern technologies. Automated refactoring ensured functional equivalence, cost reduction, and seamless integration. Crucial functionalities such as billing, invoicing, and product catalog management were streamlined. Transitioning to Java and a relational database optimized data management, while comprehensive testing ensured application reliability. Migration to AWS improved deployment flexibility, and the establishment of a CI/CD pipeline expedited releases. Future improvements include breaking down the application into microservices. Lessons learned emphasized the importance of testing, code maintenance, developer cross-training, and thorough application understanding. Overall, the project transitioned from cost-cutting to strategic enhancement, enhancing customer service and competitive advantage in the media industry.



De-risking Db2 on-prem batch

- Principles align with AWS mass migration methodology
  - Start small, Scale fast
  - Migrate then optimize
  - Automate as much as possible
  - Go build

The AWS mass migration methodology, known as the AWS Migration Acceleration Program (MAP), is a comprehensive cloud migration program designed to accelerate the migration and modernization journey to the cloud. It’s based on AWS’s experience with migrating thousands of enterprise customers to the cloud.



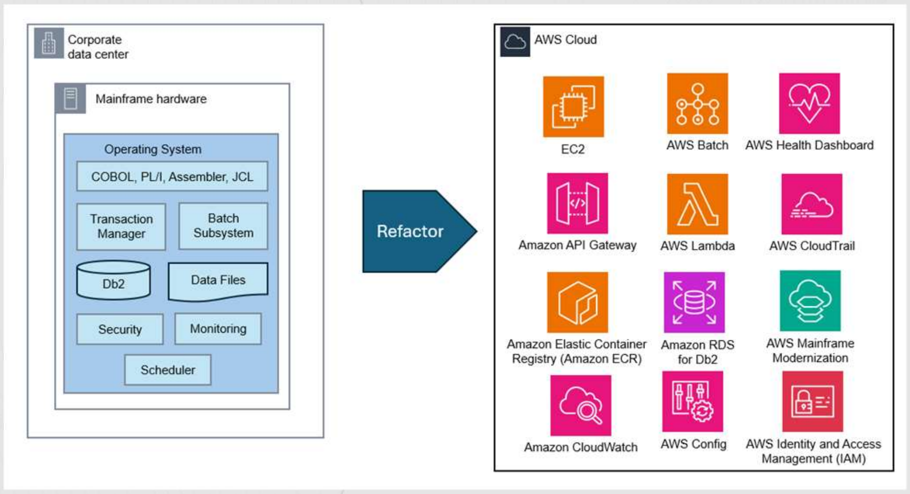
Business Benefits

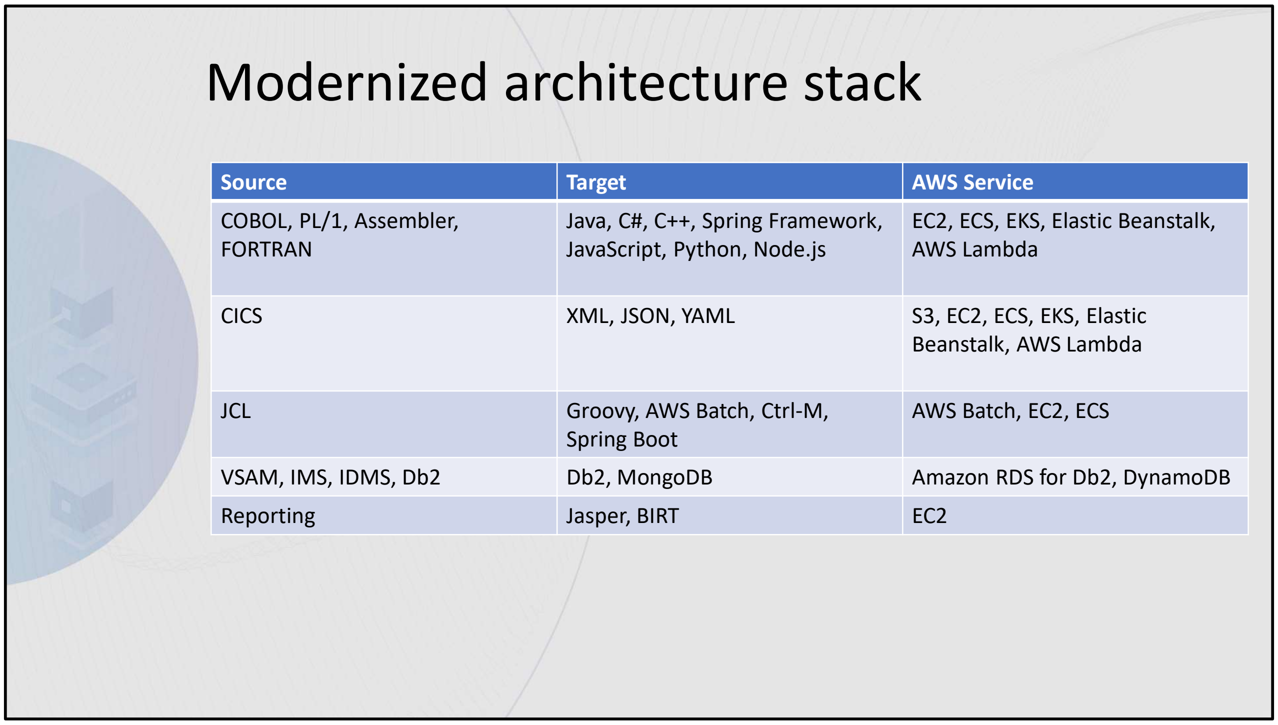
	Business Benefits	Approach
Start small	• Measurable benefits • Start with one workload • Prove feasibility and value • Enable innovation	• Break down into workloads • Migrate workloads in separate projects • Each project provides business value • Develop best practices
Migrate quickly	• De-risk short term business benefits • Infrastructure cost reduction • Infrastructure agility and elasticity • Leverage DevOps	• Automate where possible, to minimize technical risk • Automate to avoid increasing duration, cost and human risks
Optimize on AWS	• Agility with cloud-native services • Agility with polyglot architecture • Innovate	• Incremental transitions • From COBOL to Java, Python, C#, etc. • From IaaS to cloud-native • From monolith to microservices
Go Build	• Prove technical viability and deliverable quality early • Prove migration speed and functional equivalence early • Early benefits from deliverables	• Proof of Concept • Migration Plan • First workload into production

Explain slide

	Business Benefits	Approach
Start small	• Measurable benefits • Start with one workload • Prove feasibility and value • Enable innovation	• Break down into workloads • Migrate workloads in separate projects • Each project provides business value • Develop best practices
Migrate quickly	• De-risk short term business benefits • Infrastructure cost reduction • Infrastructure agility and elasticity • Leverage DevOps	• Automate where possible, to minimize technical risk • Automate to avoid increasing duration, cost and human risks
Optimize on AWS	• Agility with cloud-native services • Agility with polyglot architecture • Innovate	• Incremental transitions • From COBOL to Java, Python, C#, etc. • From IaaS to cloud-native • From monolith to microservices
Go Build	• Prove technical viability and deliverable quality early • Prove migration speed and functional equivalence early • Early benefits from deliverables	• Proof of Concept • Migration Plan • First workload into production

On-prem refactoring

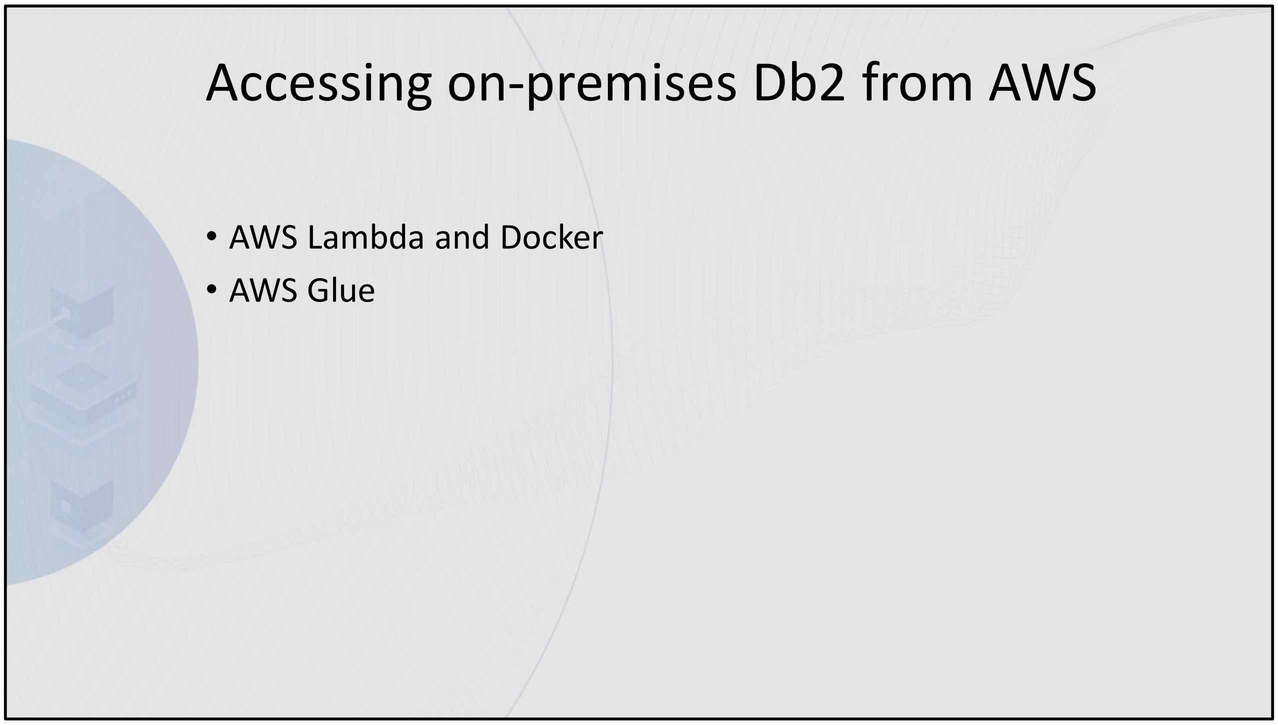




Modernized architecture stack

Source	Target	AWS Service
COBOL, PL/1, Assembler, FORTRAN	Java, C#, C++, Spring Framework, JavaScript, Python, Node.js	EC2, ECS, EKS, Elastic Beanstalk, AWS Lambda
CICS	XML, JSON, YAML	S3, EC2, ECS, EKS, Elastic Beanstalk, AWS Lambda
JCL	Groovy, AWS Batch, Ctrl-M, Spring Boot	AWS Batch, EC2, ECS
VSAM, IMS, IDMS, Db2	Db2, MongoDB	Amazon RDS for Db2, DynamoDB
Reporting	Jasper, BIRT	EC2

Explain slide



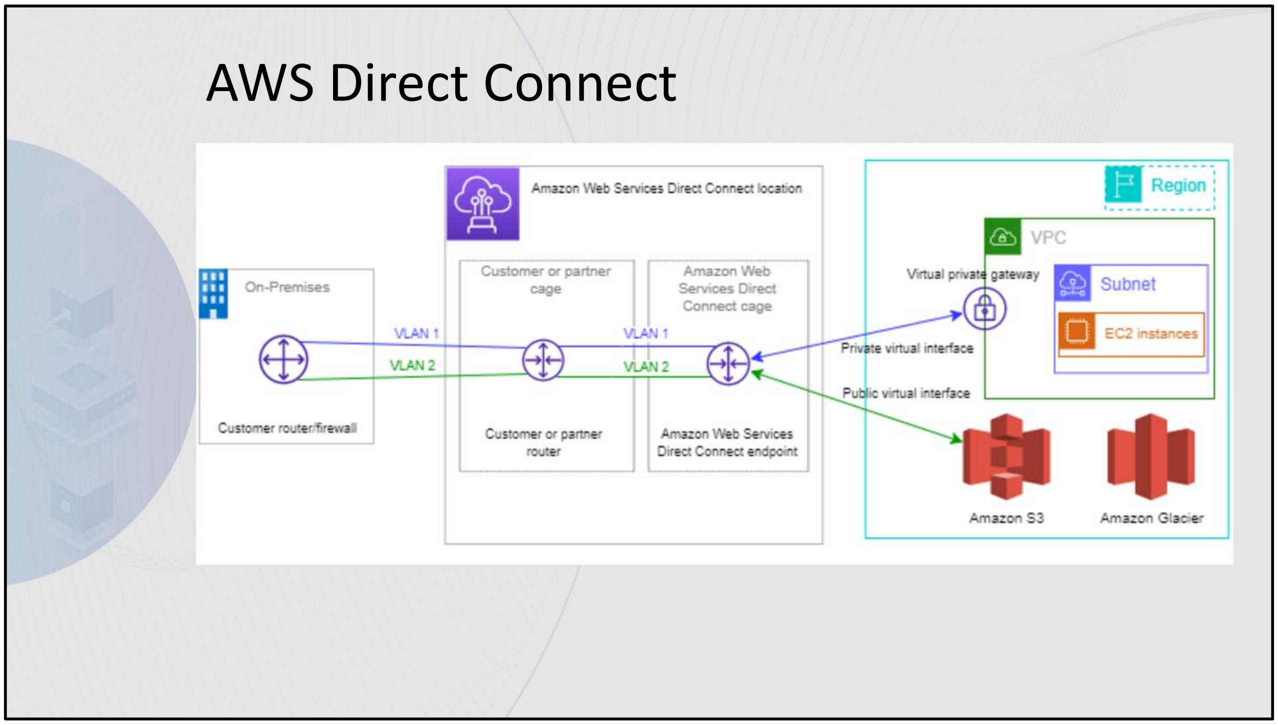
Accessing on-premises Db2 from AWS

- AWS Lambda and Docker
- AWS Glue

Accessing an on-premises DB2 database from AWS

AWS Lambda and Docker: You can use AWS Lambda to run external transactions on DB2 for IBM i databases. This involves using Docker on Amazon ECR and AWS Lambda container images to transfer data between the two environments1. The process includes building a Docker image with DB2 interfacing code, deploying it to ECR, and creating a Lambda function from the image to run your queries on the target DB2 database.

AWS Glue: AWS Glue is a fully managed ETL service that can connect to on-premises JDBC data stores, including DB2. It allows you to access and analyze data stores using JDBC connectivity through elastic network interfaces (ENIs) in an Amazon VPC. This connection can be established over a virtual private network (VPN) or AWS Direct Connect (DX)2.



AWS Direct Connect

AWS offers a connectivity solution called Direct Connect, which addresses concerns about sending large amounts of data over the public internet due to potential inconsistencies and security regulations. Direct Connect establishes a private network connection between corporate offices or traditional data centers and AWS Virtual Private Cloud (VPC) without relying on the public internet.

To set up Direct Connect, one typically engages with a partner internet service provider to establish a dedicated network link between their network and the

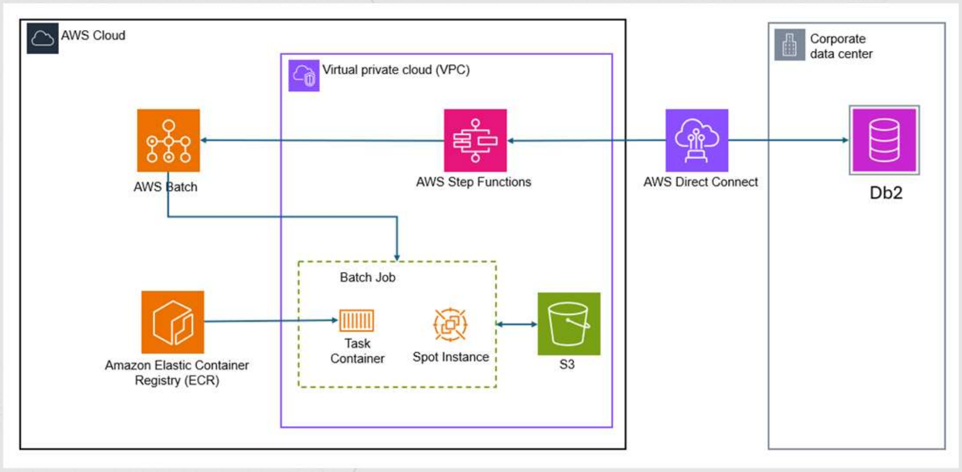
Direct Connect facility. Approval from AWS is required, which involves a brief waiting period. Once approved, a special authorization letter is provided to the network partner to establish the connection with AWS's networking equipment. This creates a private network link into AWS's backbone network.

Direct Connect offers various connection speeds, from 50 megabits per second to 100 gigabits per second, depending on the location and partners involved. Apart from the speed benefits, Direct Connect can significantly reduce data transfer costs compared to using the public internet, especially for large volumes of data. For instance, while outbound data traffic from AWS to office locations via the internet might cost nine cents per gigabyte, Direct Connect could reduce this to around two cents per gigabyte.

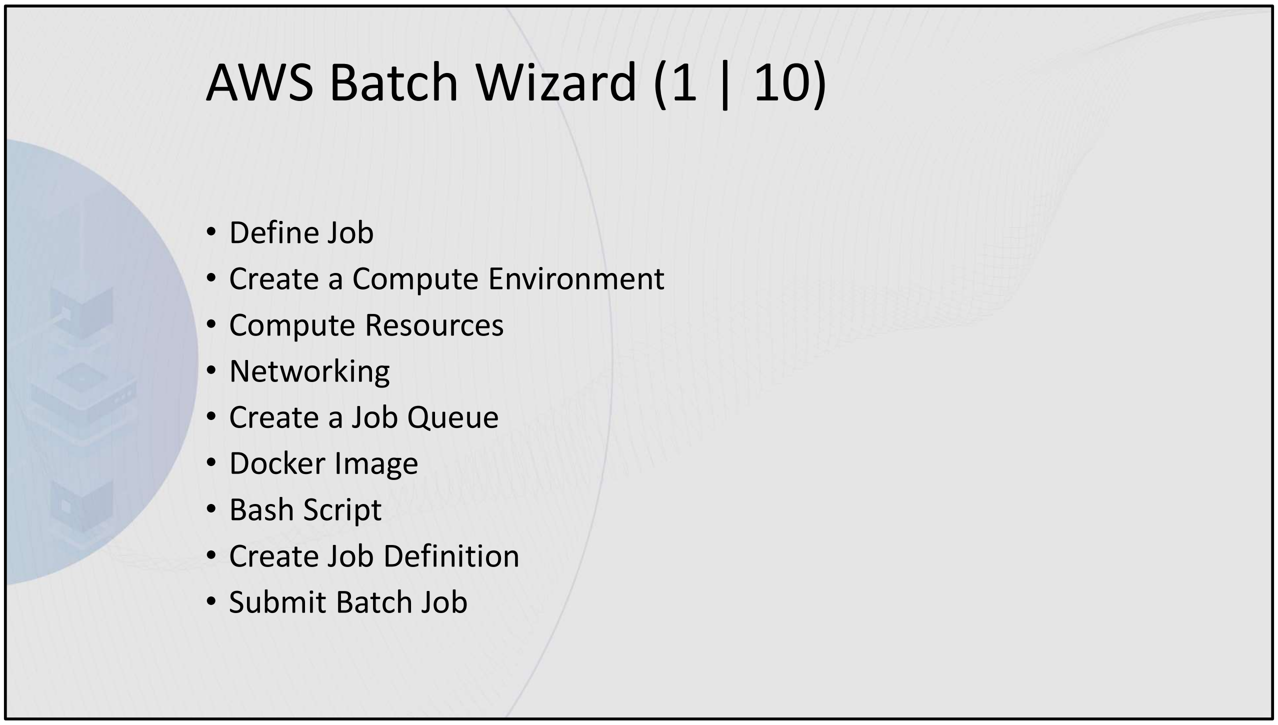
However, there are considerations and potential risks with Direct Connect. It relies on a single network connection, making it vulnerable to failures, whether from equipment malfunctions, maintenance activities, or external factors like cable damage. To mitigate these risks, redundancy is crucial, often requiring at least two Direct Connect connections to ensure continuous operation and business-critical traffic flow.

In summary, AWS Direct Connect provides a secure, high-speed, and cost-effective solution for connecting corporate offices and data centers to AWS VPC, bypassing the public internet. However, implementing redundancy is essential to safeguard against potential network failures and ensure uninterrupted connectivity.

Db2 with Direct Connect and AWS Batch



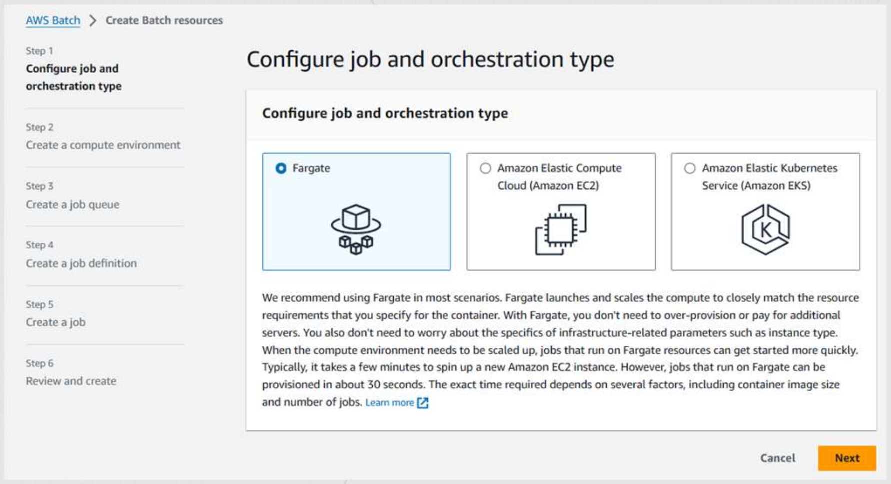
Explain slide



AWS Batch Wizard (1 | 10)

- Define Job
- Create a Compute Environment
- Compute Resources
- Networking
- Create a Job Queue
- Docker Image
- Bash Script
- Create Job Definition
- Submit Batch Job

AWS Batch Wizard (2 | 10)



AWS Batch > Create Batch resources

Step 1
Configure job and orchestration type

Step 2
Create a compute environment

Step 3
Create a job queue

Step 4
Create a job definition

Step 5
Create a job

Step 6
Review and create

Configure job and orchestration type

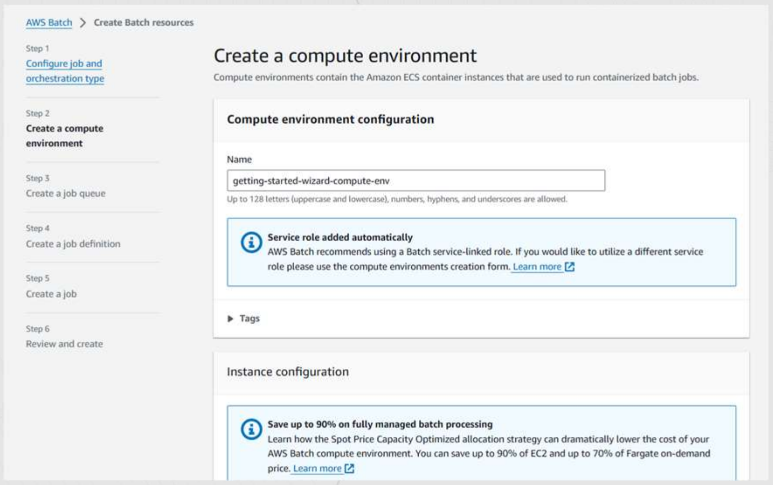
Configure job and orchestration type

[✓ Fargate] Amazon Elastic Compute Cloud (Amazon EC2) Amazon Elastic Kubernetes Service (Amazon EKS)

We recommend using Fargate in most scenarios. Fargate launches and scales the compute to closely match the resource requirements that you specify for the container. With Fargate, you don't need to over-provision or pay for additional servers. You also don't need to worry about the specifics of infrastructure-related parameters such as instance type. When the compute environment needs to be scaled up, jobs that run on Fargate resources can get started more quickly. Typically, it takes a few minutes to spin up a new Amazon EC2 instance. However, jobs that run on Fargate can be provisioned in about 30 seconds. The exact time required depends on several factors, including container image size and number of jobs. Learn more

Explain slide

AWS Batch Wizard (3 | 10)



AWS Batch > Create Batch resources

Step 1
Configure job and orchestration type

Step 2
Create a compute environment

Step 3
Create a job queue

Step 4
Create a job definition

Step 5
Create a job

Step 6
Review and create

Create a compute environment

Compute environments contain the Amazon ECS container instances that are used to run containerized batch jobs.

Compute environment configuration

Name: getting-started-wizard-compute-env

Up to 128 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed.

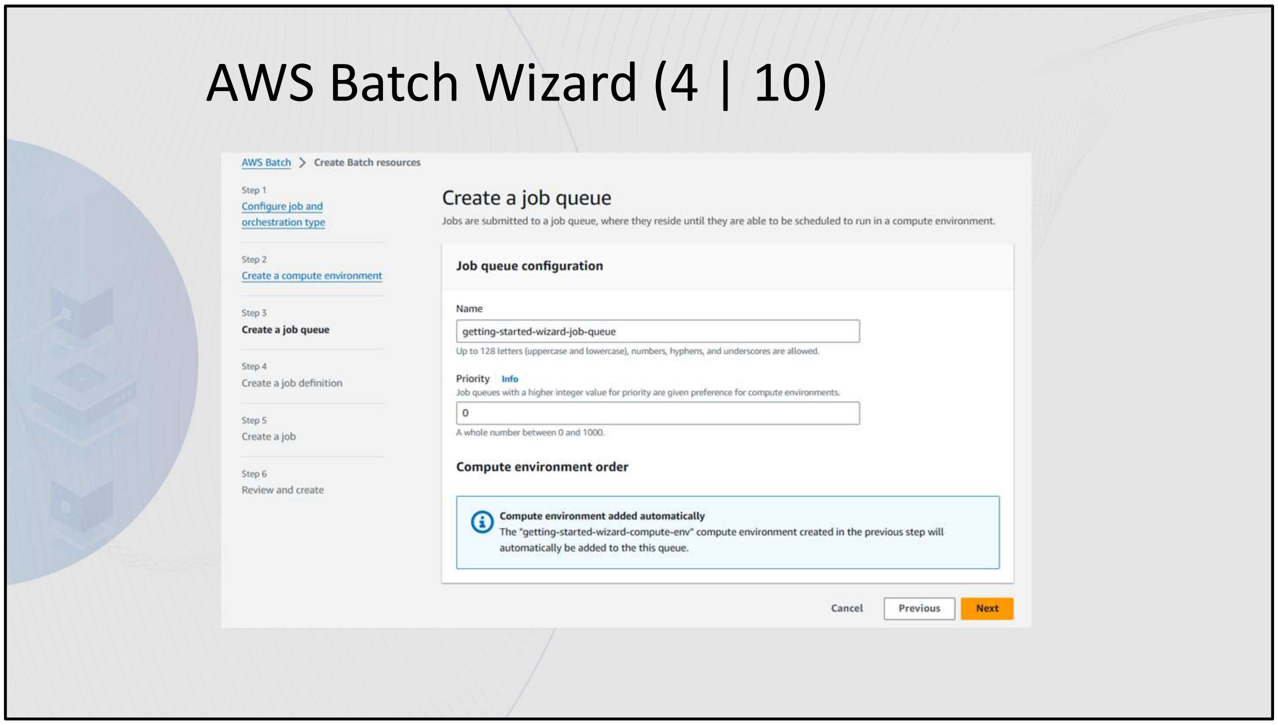
Service role added automatically
AWS Batch recommends using a Batch service-linked role. If you would like to utilize a different service role please use the compute environments creation form. Learn more

▶ Tags

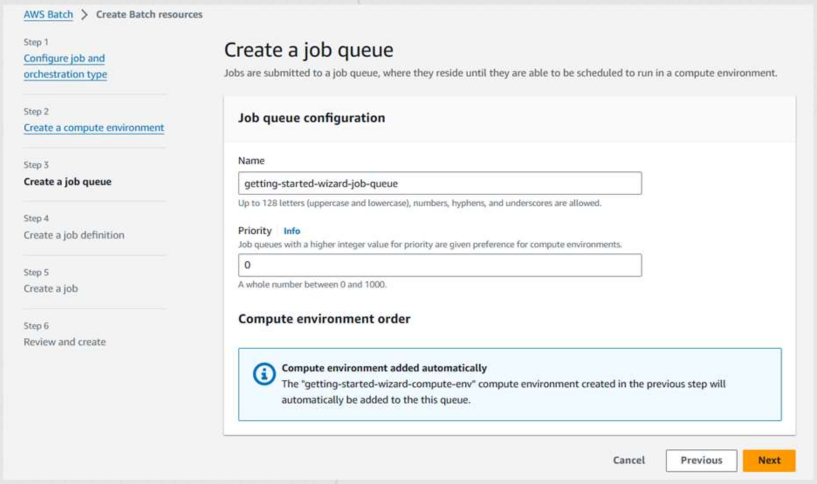
Instance configuration

Save up to 90% on fully managed batch processing
Learn how the Spot Price Capacity Optimized allocation strategy can dramatically lower the cost of your AWS Batch compute environment. You can save up to 90% of EC2 and up to 70% of Fargate on-demand price. Learn more

Explain slide



AWS Batch Wizard (4 | 10)



AWS Batch > Create Batch resources

Step 1
Configure job and orchestration type

Step 2
Create a compute environment

Step 3
Create a job queue

Step 4
Create a job definition

Step 5
Create a job

Step 6
Review and create

Create a job queue

Jobs are submitted to a job queue, where they reside until they are able to be scheduled to run in a compute environment.

Job queue configuration

Name
getting-started-wizard-job-queue
Up to 128 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed.

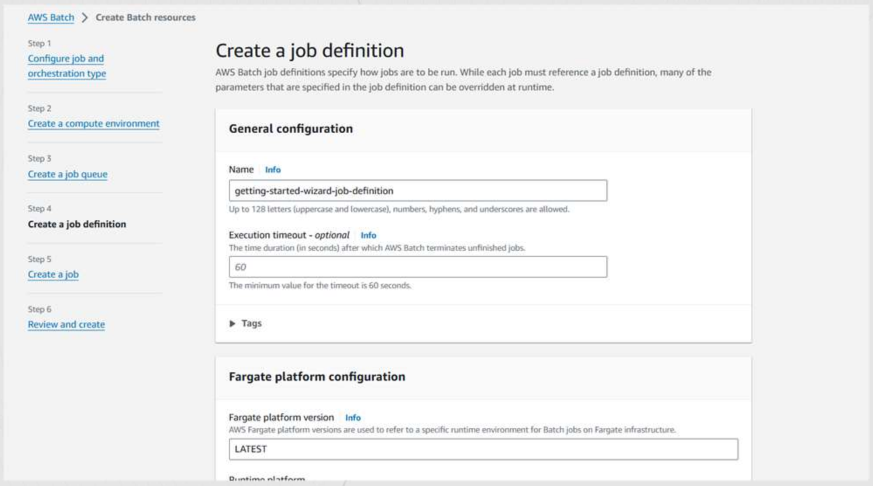
Priority Info
Job queues with a higher integer value for priority are given preference for compute environments.
0
A whole number between 0 and 1000.

Compute environment order

Compute environment added automatically
The "getting-started-wizard-compute-env" compute environment created in the previous step will automatically be added to the this queue.

Explain slide

AWS Batch Wizard (5 | 10)


AWS Batch > Create Batch resources

Step 1
Configure job and orchestration type

Step 2
Create a compute environment

Step 3
Create a job queue

Step 4
Create a job definition

Step 5
Create a job

Step 6
Review and create

Create a job definition

AWS Batch job definitions specify how jobs are to be run. While each job must reference a job definition, many of the parameters that are specified in the job definition can be overridden at runtime.

General configuration

Name Info
getting-started-wizard-job-definition
Up to 128 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed.

Execution timeout - *optional* Info
The time duration (in seconds) after which AWS Batch terminates unfinished jobs.
60
The minimum value for the timeout is 60 seconds.

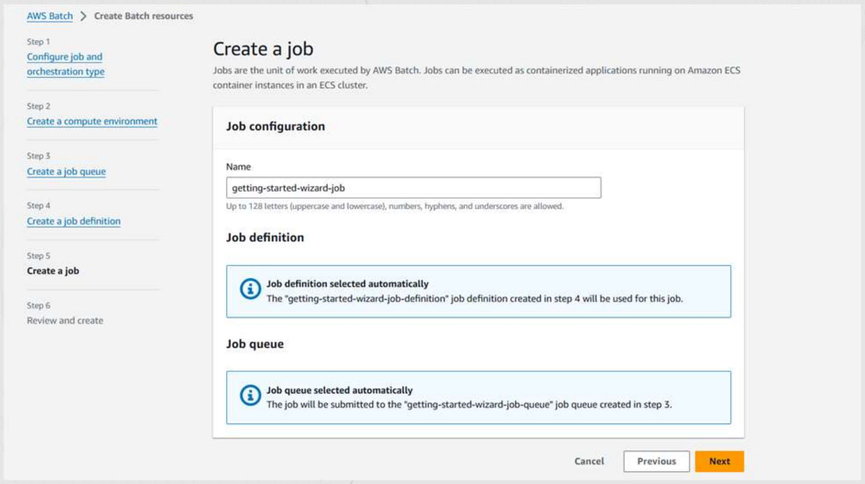
▶ Tags

Fargate platform configuration

Fargate platform version Info
AWS Fargate platform versions are used to refer to a specific runtime environment for Batch jobs on Fargate infrastructure.
LATEST

Explain slide

AWS Batch Wizard (6 | 10)



Create a job

Jobs are the unit of work executed by AWS Batch. Jobs can be executed as containerized applications running on Amazon ECS container instances in an ECS cluster.

Job configuration

Name: getting-started-wizard-job

Up to 128 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed.

Job definition

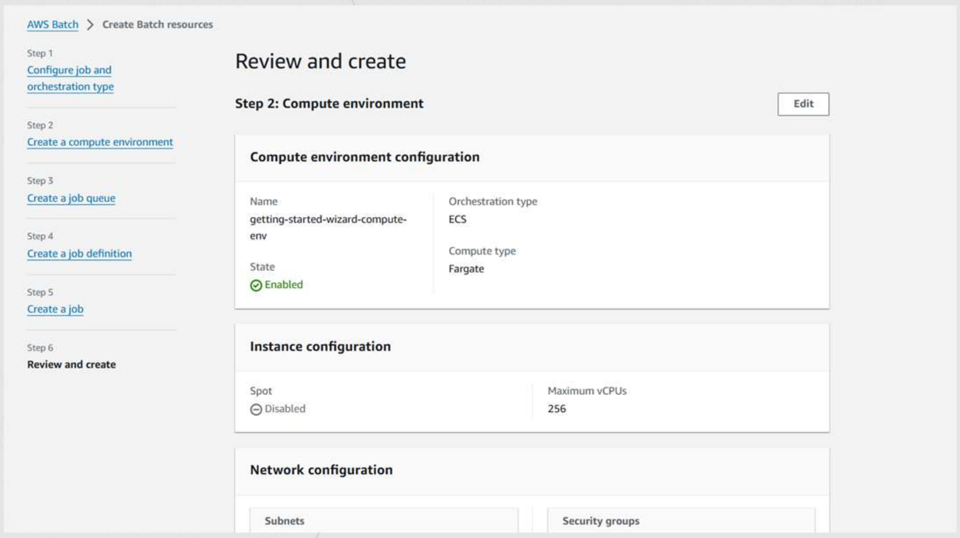
Job definition selected automatically
The "getting-started-wizard-job-definition" job definition created in step 4 will be used for this job.

Job queue

Job queue selected automatically
The job will be submitted to the "getting-started-wizard-job-queue" job queue created in step 3.

Explain slide

AWS Batch Wizard (7 | 10)



Review and create

Step 2: Compute environment

Compute environment configuration

Name: getting-started-wizard-compute-env
State: Enabled
Orchestration type: ECS
Compute type: Fargate

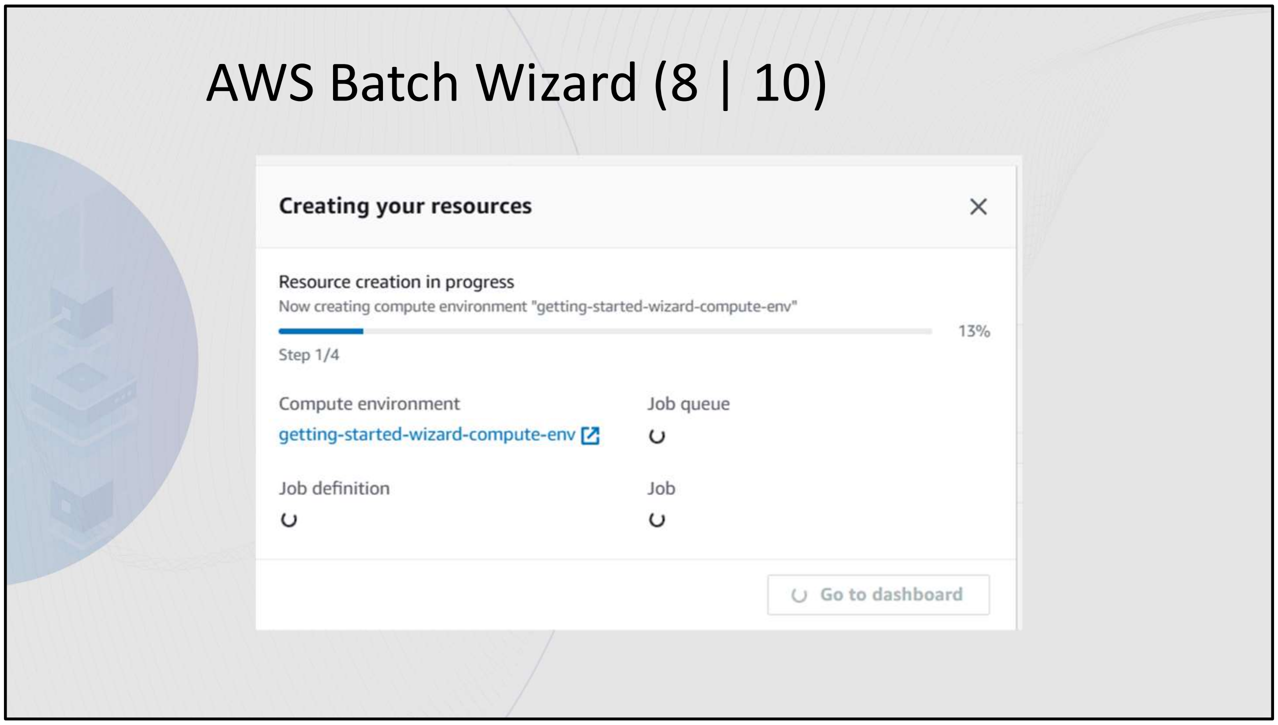
Instance configuration

Spot: Disabled
Maximum vCPUs: 256

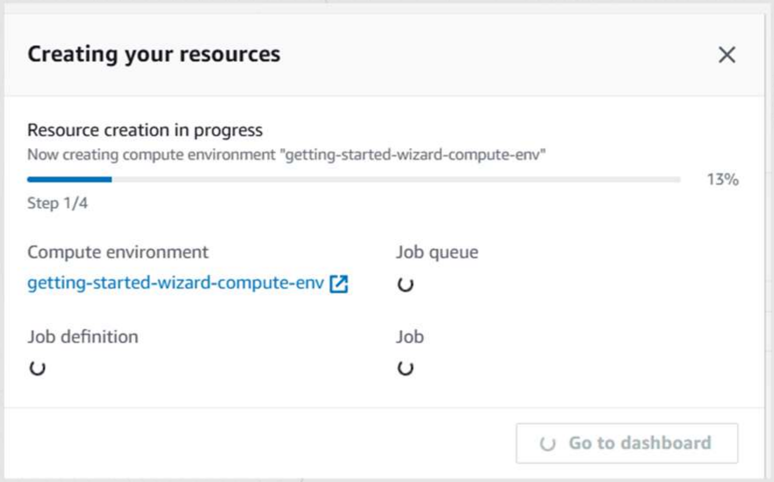
Network configuration

Subnets
Security groups

Explain slide



AWS Batch Wizard (8 | 10)



Creating your resources

Resource creation in progress

Now creating compute environment "getting-started-wizard-compute-env"

13%

Step 1/4

Compute environment: getting-started-wizard-compute-env

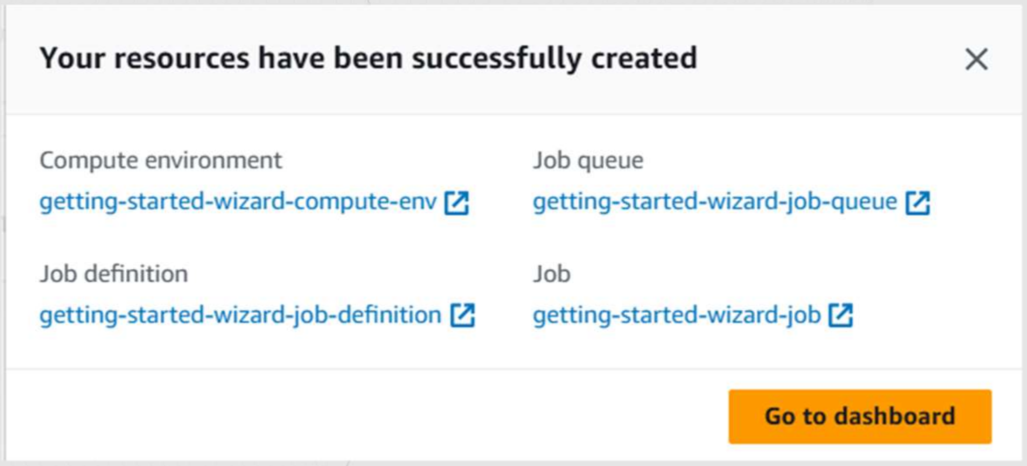
Job queue

Job definition

Job

Explain slide

AWS Batch Wizard (9 | 10)

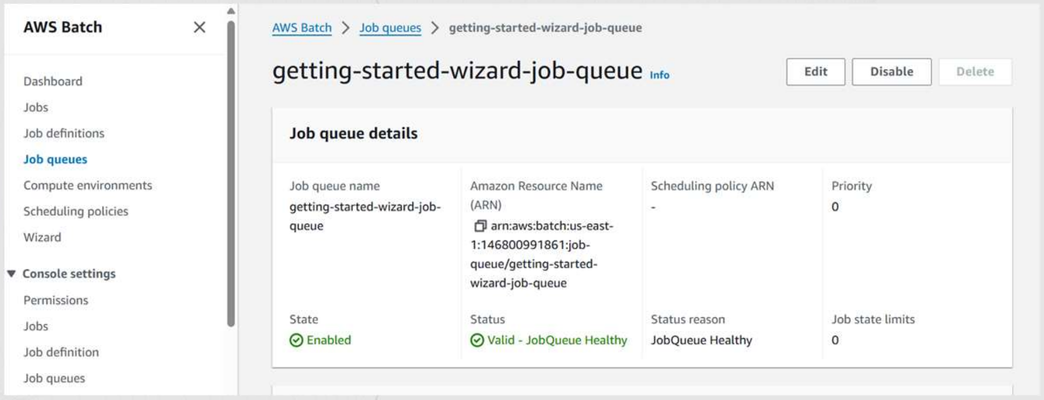


Your resources have been successfully created

Compute environment	Job queue
getting-started-wizard-compute-env	getting-started-wizard-job-queue
Job definition	Job
getting-started-wizard-job-definition	getting-started-wizard-job

Explain slide

AWS Batch Wizard (10 | 10)



getting-started-wizard-job-queue

Job queue details

Job queue name: getting-started-wizard-job-queue

Amazon Resource Name (ARN): arn:aws:batch:us-east-1:146800991861:job-queue/getting-started-wizard-job-queue

Scheduling policy ARN: -

Priority: 0

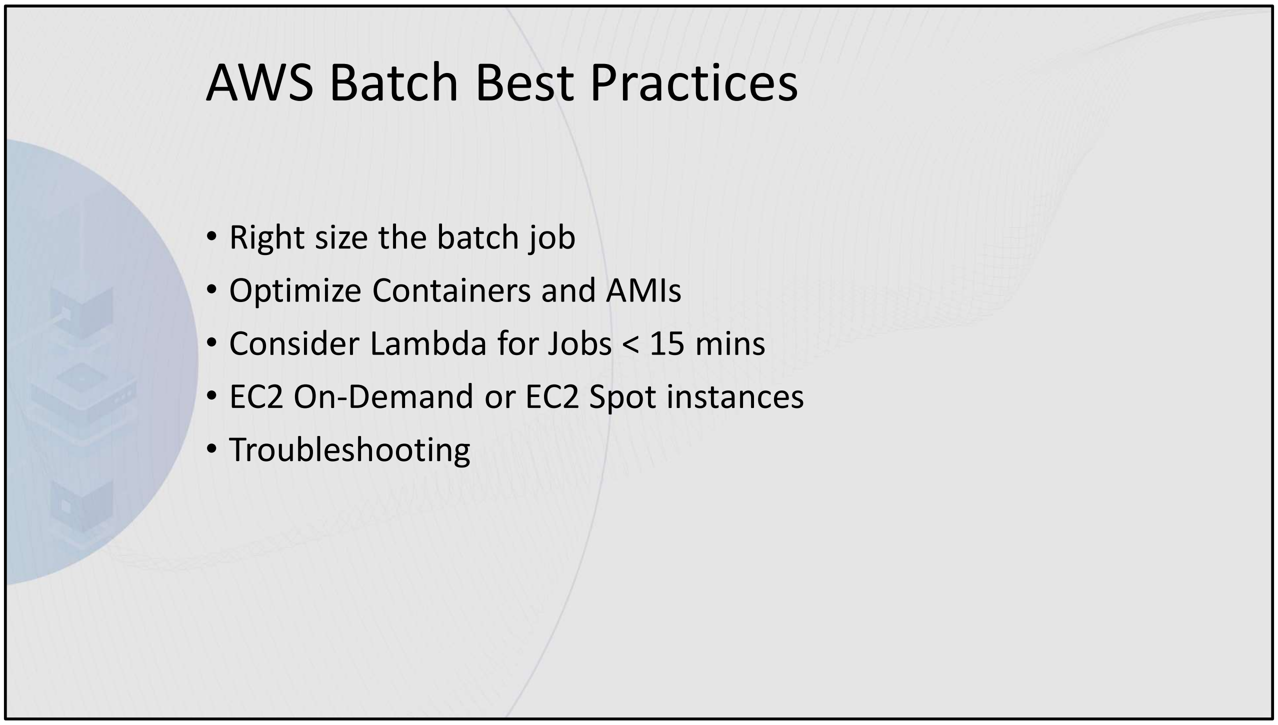
State: Enabled

Status: Valid - JobQueue Healthy

Status reason: JobQueue Healthy

Job state limits: 0

Explain slide



AWS Batch Best Practices

- Right size the batch job
- Optimize Containers and AMIs
- Consider Lambda for Jobs < 15 mins
- EC2 On-Demand or EC2 Spot instances
- Troubleshooting

Right size the batch job

AWS Batch is suitable for running jobs at scale and low cost. However, consider the following scenarios:

Short Jobs: If a job runs for only a few seconds, binpack tasks together before submitting them to AWS Batch.

Jobs That Must Run Immediately: If you need a response in under a few seconds, consider using Amazon ECS or Amazon EKS instead.

Before running a large workload (50,000 or more vCPUs), follow this checklist:

Check EC2 Quotas: Verify your Amazon EC2 quotas (limits) for both Amazon EC2 Spot and On-Demand instances.

Elastic Block Store (EBS) Quota: Ensure you have enough EBS volume quota for each Region.

Use Amazon S3 for Storage: Leverage Amazon S3 for high throughput and storage needs.

Optimize Containers and AMIs:

Choose efficient container images and optimize your Amazon Machine Images (AMIs) for performance.

Compute Environment Resource:

Decide between Amazon EC2 On-Demand or Amazon EC2 Spot instances based on your workload requirements.

Consider Lambda for Jobs < 15 mins

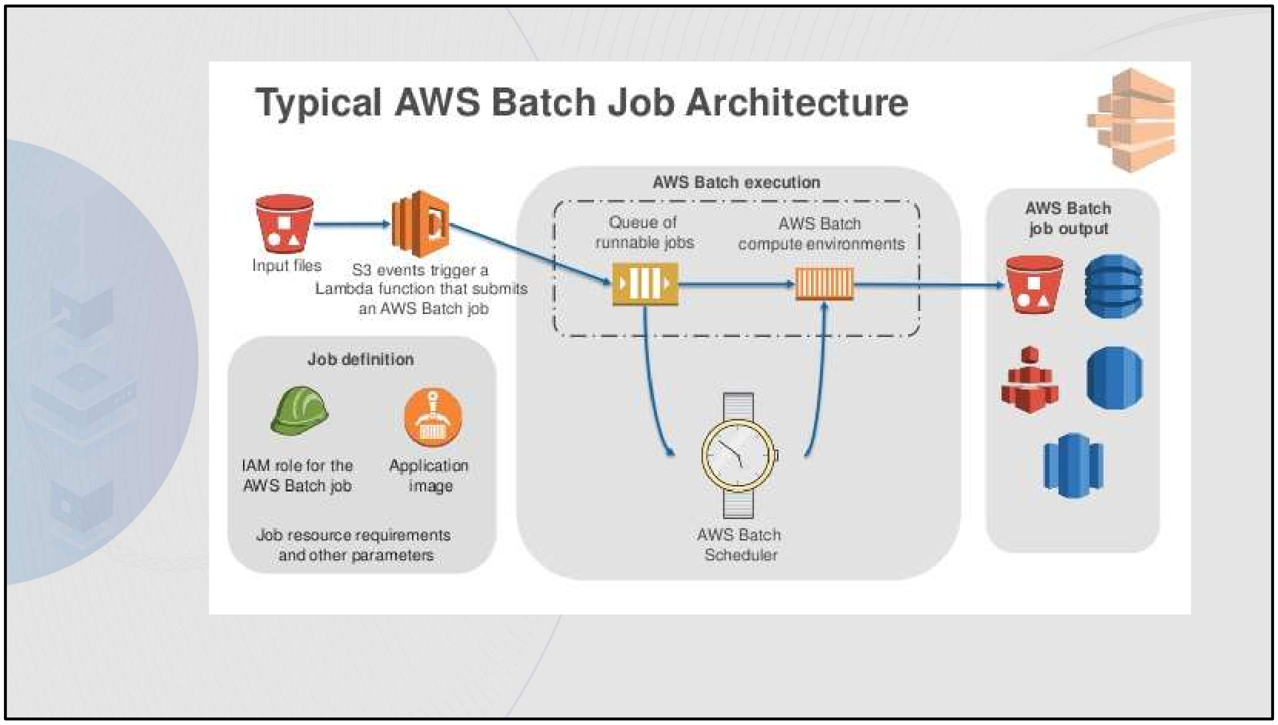
Use Amazon EC2 Spot Best Practices for AWS Batch:

Configure your job definitions to use Spot instances effectively.

Set up fallback strategies to handle Spot instance interruptions.

Common Errors and Troubleshooting:

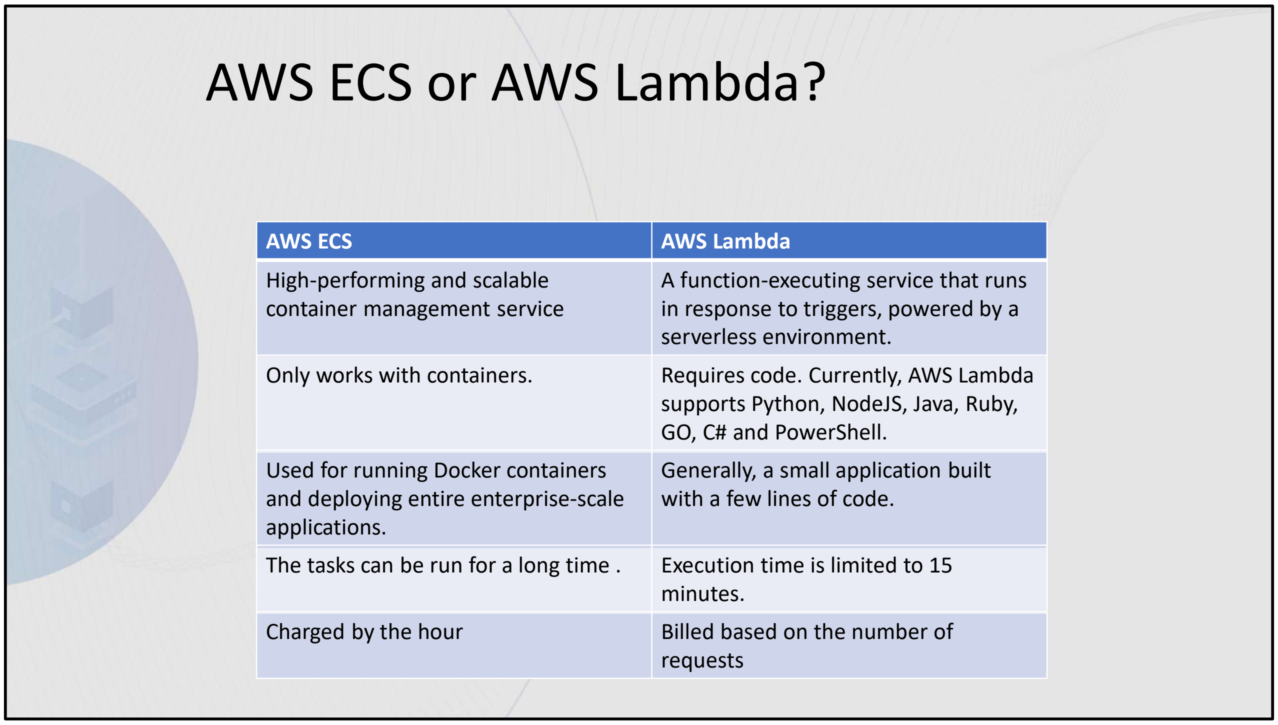
Be aware of common issues related to job scheduling, dependencies, and resource allocation.



Typical AWS Batch Job Architecture

Input files
S3 events trigger a Lambda function that submits an AWS Batch job
AWS Batch execution
Queue of runnable jobs
AWS Batch compute environments
AWS Batch Scheduler
AWS Batch job output
Job definition
IAM role for the AWS Batch job
Application image
Job resource requirements and other parameters

Explain slide



AWS ECS or AWS Lambda?

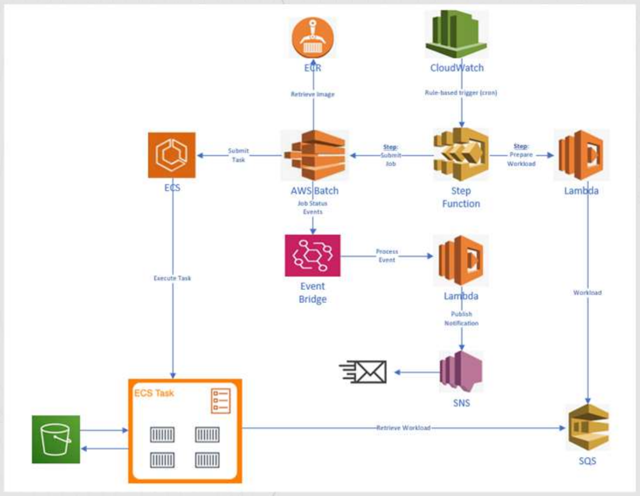
AWS ECS	AWS Lambda
High-performing and scalable container management service	A function-executing service that runs in response to triggers, powered by a serverless environment.
Only works with containers.	Requires code. Currently, AWS Lambda supports Python, NodeJS, Java, Ruby, GO, C# and PowerShell.
Used for running Docker containers and deploying entire enterprise-scale applications.	Generally, a small application built with a few lines of code.
The tasks can be run for a long time .	Execution time is limited to 15 minutes.
Charged by the hour	Billed based on the number of requests

Explain slide

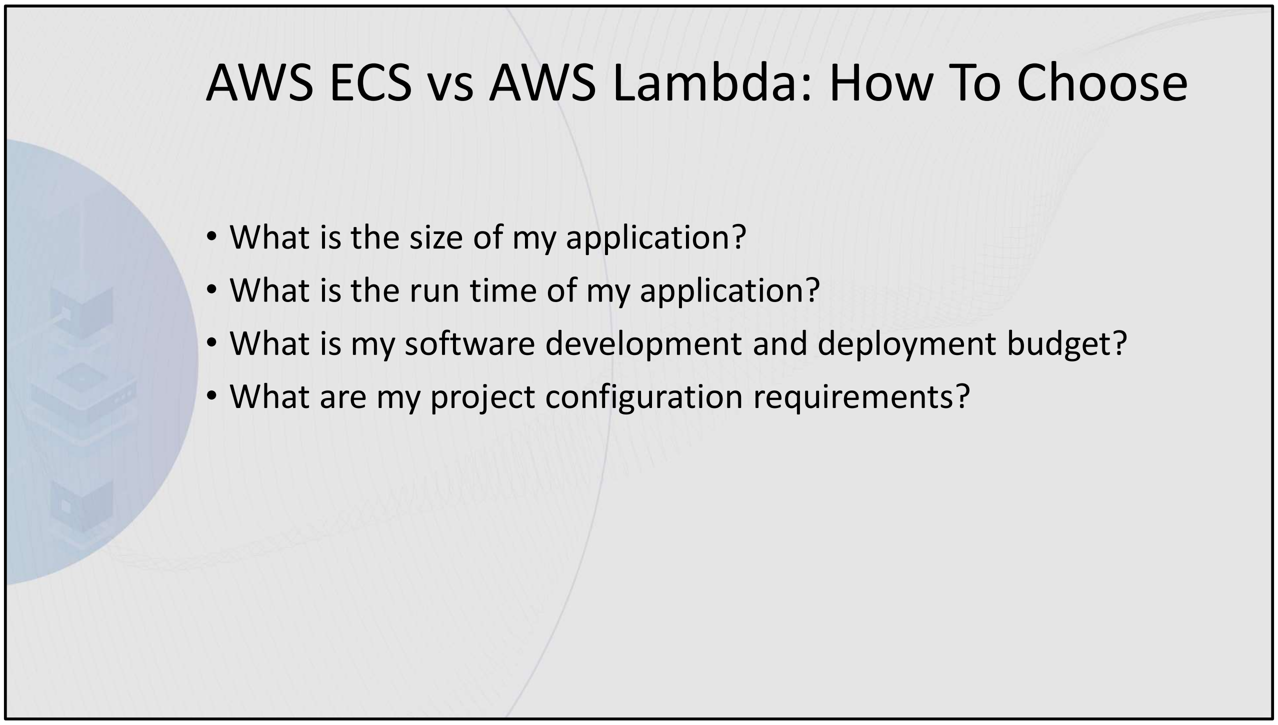
Amazon Elastic Container Service (ECS) is a fully managed container orchestration service provided by **Amazon Web Services (AWS)**. It simplifies the deployment, scaling, and management of containerized applications, making it easier for developers to build, package, and run applications using containers

AWS Lambda is an event-driven, serverless **Function as a Service (FaaS)** provided by **Amazon Web Services (AWS)**. It allows developers to run code without provisioning or managing servers. When a Lambda function is triggered, AWS automatically provisions the necessary resources to execute the code

AWS Batch Architecture with Lambda



Explain slide, defs for each component



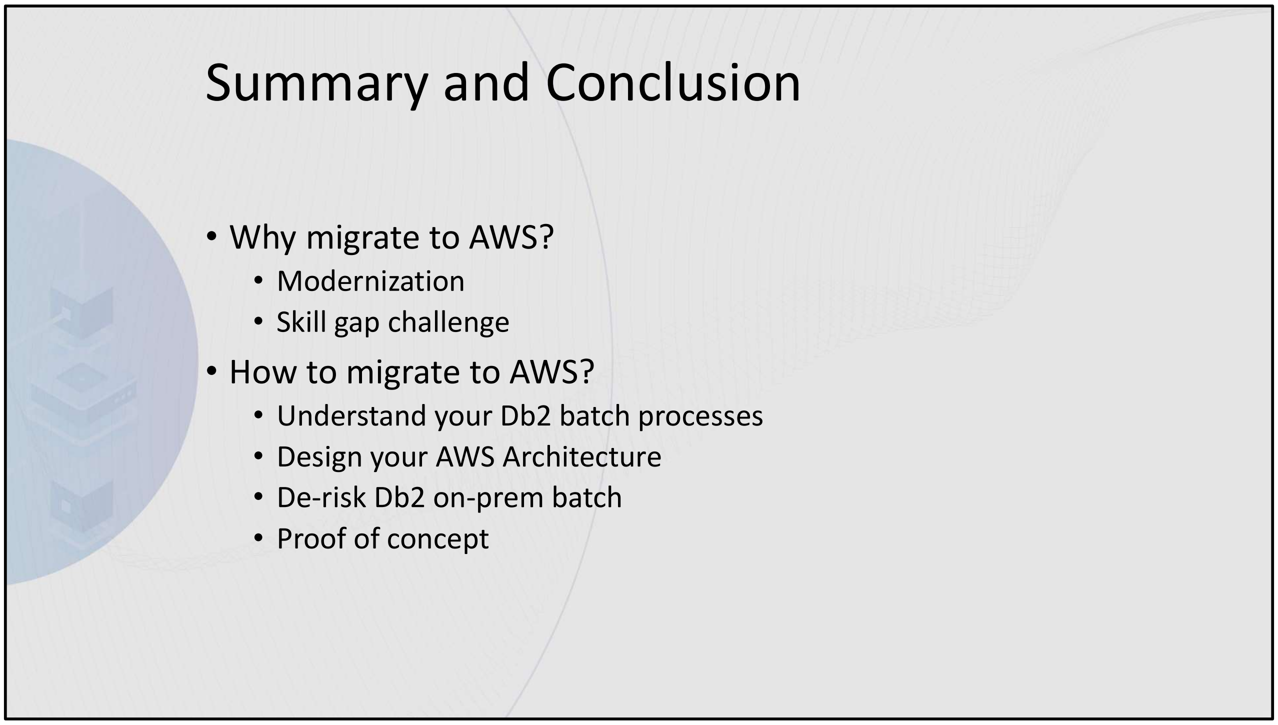
AWS ECS vs AWS Lambda: How To Choose

- What is the size of my application?
- What is the run time of my application?
- What is my software development and deployment budget?
- What are my project configuration requirements?

Choosing between AWS ECS and AWS Lambda depends on the specific needs of your application. Here’s a brief comparison to help you decide:

AWS ECS:
- Suitable for container management and orchestration.
- Offers more control over the environment and networking.
- Ideal for long-running applications and complex microservices architectures

AWS Lambda
- Serverless compute service, automatically managing the compute resources.
- Best for event-driven applications and executing code in response to triggers.
- Simplifies deployment and scaling, with a pay-per-use pricing model



Summary and Conclusion

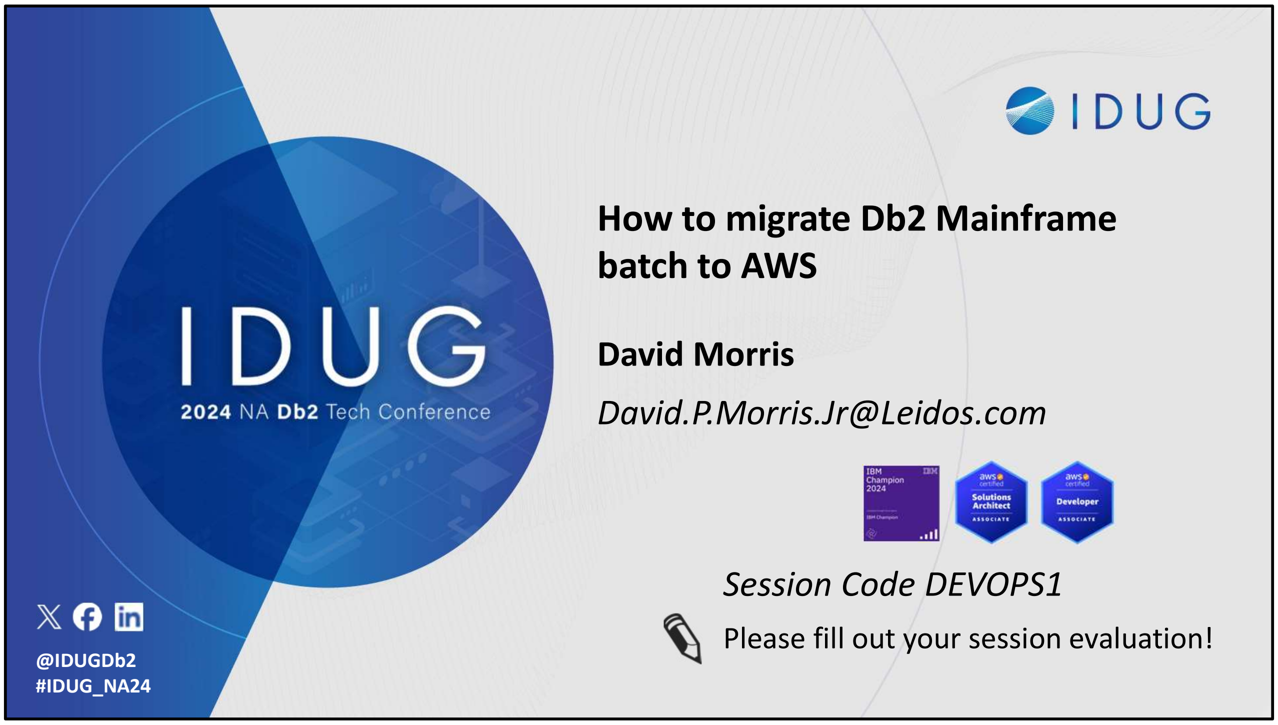
- Why migrate to AWS?
  - Modernization
  - Skill gap challenge
- How to migrate to AWS?
  - Understand your Db2 batch processes
  - Design your AWS Architecture
  - De-risk Db2 on-prem batch
  - Proof of concept

Today's summary and conclusion highlight the benefits and key steps involved in migrating to AWS, emphasizing its advantages such as cost savings, scalability, and modernization opportunities.

Migrating to AWS offers significant benefits, including cost savings through its pay-as-you-go model, agility for quick deployment and innovation, and the opportunity to modernize applications and infrastructure. It also helps mitigate skill gaps associated with on-premises mainframe Db2 batch processes.

The migration process involves several key steps to ensure a smooth transition. First, understanding Db2 batch processes is crucial for analyzing current workflows and requirements. Then, designing an AWS architecture that is scalable, secure, and efficient is essential, leveraging best practices and reference architectures provided by AWS. De-risking Db2 on-prem batch processes involves evaluating potential risks and developing mitigation strategies before migration. Finally, conducting a proof of concept allows for testing the migration strategy to ensure it aligns with business requirements and is cost-effective.

In conclusion, migrating to AWS offers numerous benefits and opportunities for organizations, but it requires careful planning and execution. By following the outlined steps and leveraging AWS's resources, businesses can achieve a successful transition to the cloud and realize its full potential.



IDUG

IDUG

2024 NA **Db2** Tech Conference

How to migrate Db2 Mainframe batch to AWS

David Morris

David.P.Morris.Jr@Leidos.com

Session Code DEVOPS1

Please fill out your session evaluation!

@IDUGDb2
#IDUG_NA24

Thank you for attending today's presentation. I hope you found it interesting and informative on migrating Db2 Mainframe batch to AWS. If you have any questions you can reach me at my email address. I wish you best of luck slaying your IT dragons.

Please remember to fill out your session cards. The Session Code is DEVOPS1.