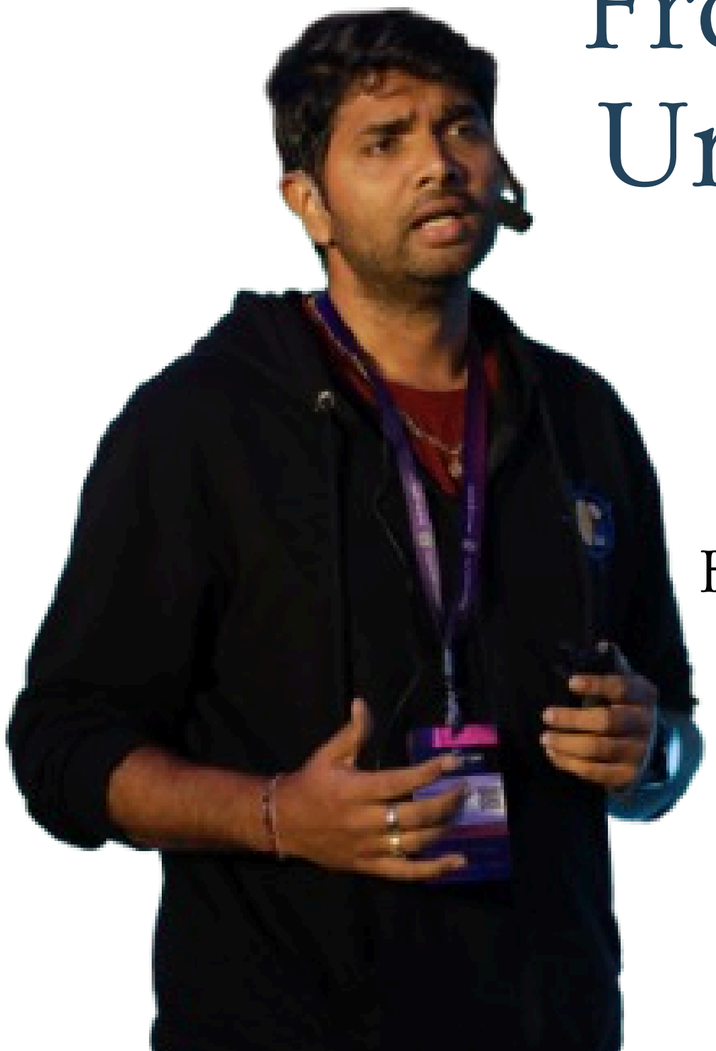


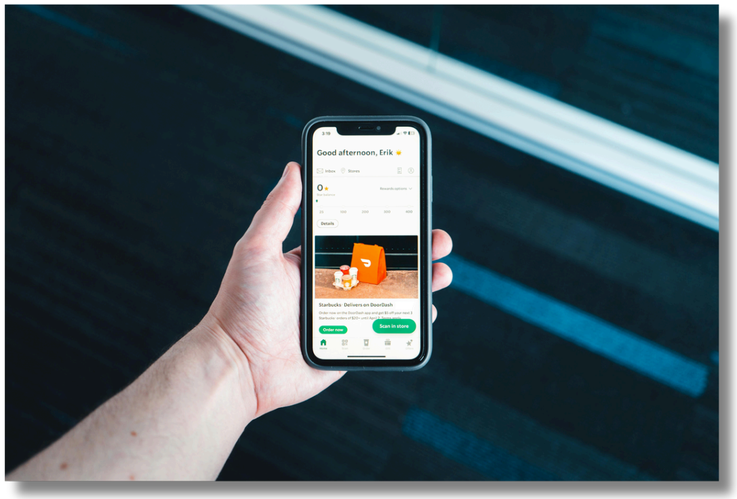
From Code to Containers: Understanding Serverless Beyond Lambda

Avinash Dalvi
Head of technology, Nushift Technologies
AWS User group Leader, Bengaluru
AWS Community Builder

It's Friday night, you and your friends want pizza...

Option 1: Make it yourself

Option 2: Just order pizza



Which would you choose? 🤔

Traditional Servers (EC2)

- ✓ You manage the "kitchen" (server)
- ✓ You maintain the "oven" (patching, updates)
- ✓ You pay for it 24/7 (idle or busy)
- ✓ Full control, full responsibility

Serverless

- ✓ Just bring your code (order pizza)
- ✓ Specify requirements (toppings, size, base)
- ✓ AWS handles everything else
- ✓ Pay only for execution time**

How do you "order pizza" in AWS? 🍕

Serverless ≠ No Servers

Serverless = You don't worry about servers!

Ordering Your Serverless Pizza

BASE(RUNTIME/LANGUAGE)

- Python, Node.js, Java, Go, .NET, Ruby...
- Or bring your own custom base (containers)

TOPPINGS (RESOURCES)

- RAM: 128MB to 10GB+
- Storage: Temporary disk space
- CPU: Scales with memory

SIZE (CODE COMPLEXITY)

- Small: Simple functions
- Medium: Moderate logic
- Large: Complex applications

WHEN SHOULD IT RUN?

- Immediate (event-triggered
- Scheduled (cron jobs)
- On-demand (API calls)

The Serverless Menu

Both are "ordering pizza" but different options...

Different options...

Aspect	Lambda	Fargate
What it is	Standard menu	Custom recipe
Base options	Pre-set runtimes	Any container image
Customization	Limited to menu	Fully customizable
Execution time	Max 15 minutes	Unlimited
Code size	250MB unzipped	No limit
Use case	Quick functions	Long processes
Cold starts	Sometimes	More consistent

Basic settings Info

Description - *optional*

Memory Info

Your function is allocated CPU proportional to the memory configured.

128 MB

Set memory to between 128 MB and 10240 MB

Ephemeral storage Info

You can configure up to 10 GB of ephemeral storage (/tmp) for your function. View pricing

512 MB

Set ephemeral storage (/tmp) to between 512 MB and 10240 MB.

SnapStart Info

Reduce startup time by having Lambda cache a snapshot of your function after the function has initialized. To evaluate whether your function code is resilient to snapshot operations, review the SnapStart compatibility considerations. For Python and .NET runtimes, view pricing.

None

Supported runtimes: .NET 8 (C#/F#/PowerShell), Java 11, Java 17, Java 21, Python 3.12, Python 3.13.

Timeout

1 min 0 sec

Execution role

Choose a role that defines the permissions of your function. To create a custom role, go to the IAM console.

- (•) Use an existing role
- ( ) Create a new role from AWS policy templates

Existing role

Choose an existing role that you've created to be used with this Lambda function. The role must have permission to upload logs to Amazon CloudWatch Logs.

LambdaChime

View the LambdaChime role on the IAM console.

Image Resizer

```python
# Your code (the pizza recipe)
import boto3
from PIL import Image

def lambda_handler(event, context):
    # Get uploaded image from S3
    bucket = event['Records'][0]['s3']['bucket']['name']
    key = event['Records'][0]['s3']['object']['key']

    # Resize image
    image = Image.open(f'/tmp/{key}')
    image.thumbnail((200, 200))

    # Save thumbnail
    image.save(f'/tmp/thumb_{key}')
    return {'status': 'success'}
```

When Lambda Hits the Wall

- ✅ Upload video → Lambda works great!
- ✅ Generate thumbnail → Lambda perfect!
- ❌ Transcode video → Lambda fails!

Why?

- Video processing takes 45 minutes
- Lambda timeout: 15 minutes
- Need more control over environment
- Larger dependencies required

Fargate - Bring Your Own Recipe

- **You bring a container (Docker image)**
- **AWS runs it without managing servers**

Fargate - The Freedom

MORE CONTROL

- Any programming language
- Any runtime version
- Custom OS dependencies
- Specific tools/libraries

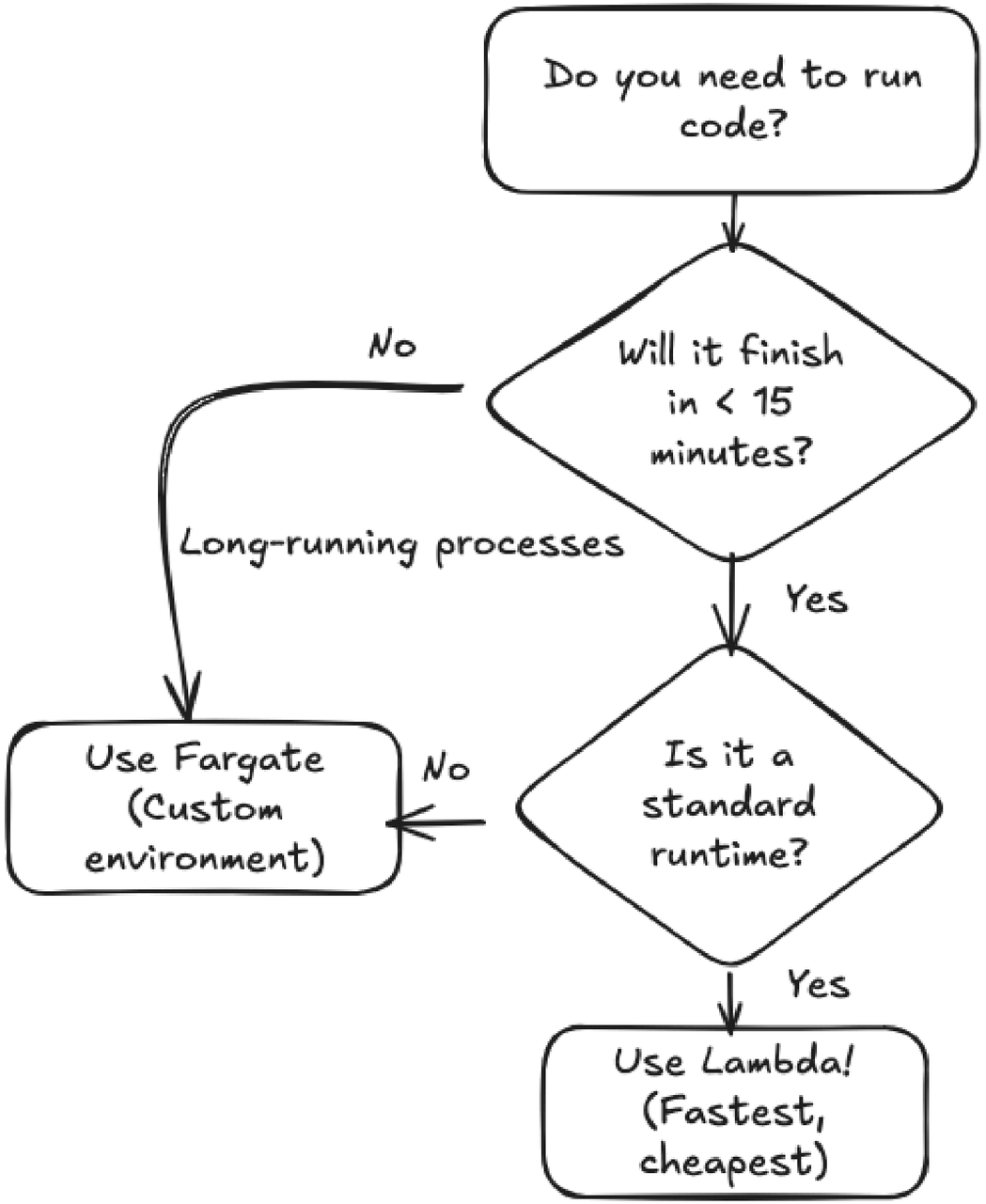
MORE CAPACITY

- Run for hours/days
- No 15-minute limit
- Up to 120GB RAM
- 4 vCPUs

MORE USE CASES

- Long-running processes
- Legacy applications
- Microservices
- Batch jobs and More...

Lambda vs Fargate - Decision Tree



Real Student Projects

ASSIGNMENT SUBMISSION PORTAL

- API endpoints (submit, list assignments)
- Email notifications
- File validation
- Cost: $0.50/month for 500 students

CODE PLAGIARISM CHECKER

- Lambda: Receive submissions, trigger checks
- Fargate: Run complex similarity algorithms (30 min)
- Lambda: Send results email
- Cost: $2/month for 200 submissions

LIVE STREAMING PLATFORM

- 24/7 streaming server
- Custom RTMP processing
- Could not use Lambda (continuous running)

Architecture Patterns to Try

- Event-Driven API (S3 upload → Lambda → DynamoDB)
- Scheduled Jobs (EventBridge (cron) → Lambda → Process data)
- Microservices (API Gateway → Lambda/Fargate → Services)
- Data Pipeline (S3 → Lambda (trigger) → Fargate (process) → S3)
- Webhook Handler (GitHub/Stripe → API Gateway → Lambda → Action)

Start small, scale infinitely!

The Serverless Mindset Shift

“I need a server to run my code”

- Provision capacity for peak load
- Manage infrastructure
- Pay for idle time
- Scale manually

The Serverless Mindset Shift

“I need to run code when X happens”

- Specify requirements
- Let AWS handle infrastructure
- Pay for execution only
- Scale automatically

Think in events, not servers!

When NOT to Use Serverless

“Be honest – serverless isn't always the answer”

- Predictable (and high) traffic
- When you need more control
- Long-running tasks
- When you aren't sure of your requirements

Key Takeaways

- Serverless = Ordering Pizza
- Lambda vs Fargate
- Focus on code, not servers
- Pay for value, not idle time

References

- https://www.ben-morris.com/when-to-use-serverless-architecture-and-when-not-to/
- https://aws.amazon.com/blogs/compute/operating-lambda-anti-patterns-in-event-driven-architectures-part-3/
- https://aws.amazon.com/blogs/architecture/mistakes-to-avoid-when-implementing-serverless-architecture-with-lambda/

Thank you!

Just search