AWS Serverless Fundamentals

Core Benefits of Serverless Services

- **No Server Management**: AWS handles infrastructure completely
- **Continuous Scaling**: Automatic resource adjustment
- **Pay-for-Value Services**: Cost based on actual usage
- **Built-in High Availability**: Fault-tolerant design
- **Ideal for**: Event-driven and microservice architectures

Origin and Philosophy

- Introduced in 2013 when AWS noticed customers using EC2 instances for very short periods
- Designed to optimize resource efficiency
- Enables deployment with minimal infrastructure overhead

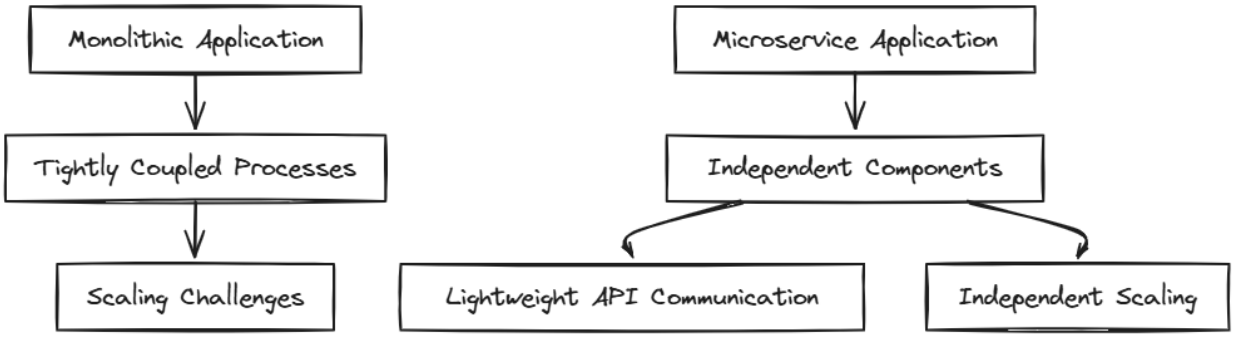
Microservices Architecture

Key Benefits of Microservices

1. **Team Agility**
    - Enables small, independent teams
    - Accelerates development cycles
2. **Code Reusability**
    - Services can be repurposed across different applications
    - Reduces redundant coding efforts
3. **Flexible Scaling**
    - Independent scaling of individual services
    - Precise infrastructure and cost management
4. **Technological Freedom**
    - Teams can choose optimal tools for specific challenges
    - Breaks "one-size-fits-all" architectural constraints

5. **Resilience**
    - Partial system failure doesn't crash entire application
    - Improved system stability
6. **Simplified Deployment**
    - Supports continuous integration/delivery
    - Encourages experimentation

Microservices vs Monolithic Architecture



AWS Lambda: Serverless Compute Service

Key Characteristics

- Run code without server provisioning
- Configurable runtime language
- Maximum function duration: 15 minutes
- Memory allocation range: 128MB - 10,240MB

Deployment Options

- .zip file archives
- Container images

Invocation Scenarios

1. **Synchronous Processing**
    - Web applications
    - Microservices
    - Machine learning inferences
2. **Asynchronous Processing**
    - Scheduled events
    - Queued message handling
3. **Streaming Processing**
    - Continuous data streams
    - Data analytics transformations

Container Technologies

Container vs Virtual Machines

Feature	Virtual Machines	Containers
OS Layer	Full Guest OS	Shared Host OS
Resource Usage	High	Lightweight
Startup Speed	Slow	Rapid
Scalability	Limited	Highly Scalable

Container Use Cases

- Microservice applications
- Batch processing
- Machine learning model scaling
- Hybrid architecture standardization
- Cloud migration

AWS Container Services

Container Registry

- **Amazon Elastic Container Registry (ECR)**
    - Secure, scalable image storage
    - Supports private and public repositories

Orchestration Options

1. **Amazon Elastic Container Service (ECS)**
    - AWS-native container management
    - Integrated with AWS tools
2. **Amazon Elastic Kubernetes Service (EKS)**
    - Kubernetes-based container orchestration
    - Suitable for existing Kubernetes environments

Compute Platforms

- Amazon EC2
- AWS Fargate
- AWS Lambda (limited container support)

Architectural Patterns

Serverless Microservice Patterns

1. **RESTful APIs**
    - Stateless communication
    - Amazon API Gateway + AWS Lambda
2. **Container-Based**
    - Long-running processes
    - AWS Fargate + Application Load Balancer

3. **Streaming Services**
    - Event-driven architectures
    - AWS Lambda + Amazon Kinesis

Advanced Considerations

Lambda Function Limitations

- Maximum runtime: 15 minutes
- Memory limit: 10 GB
- Recommended for short, event-driven tasks

Cost Optimization Strategies

- Compare pricing between Lambda, ECS, Fargate
- Consider transaction volume and runtime duration

Key Takeaways

- Serverless enables rapid, efficient application development
- Microservices offer flexibility and resilience
- AWS provides comprehensive tools for modern cloud architectures