COMP5349– Cloud Computing
Week 11: Serverless Architecture

Dr. Ying Zhou

The University of Sydney

Table of Contents

01 Function as a Service

02 Amazon Serverless Computing

03 Lambda Execution Environment

04 Lambda Execution Anti-Patterns

05 Amazon API Gateway

Function as a Service

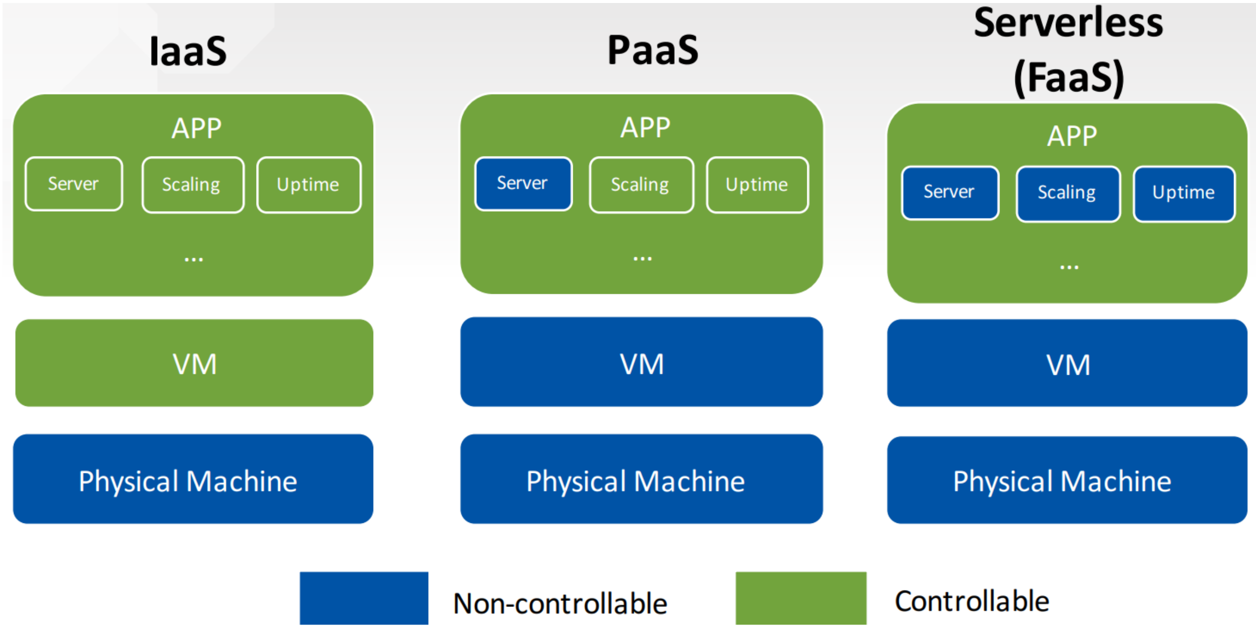
A Berkeley View on Cloud Computing

- In 2009, a group of UC Berkeley researchers published a seminal paper "The Berkeley View on Cloud Computing"
- The paper identified various **XaaS** models and highlighted two competing approaches to virtualization in the cloud
  - "Amazon EC2 is at one end of the spectrum. An EC2 instance looks much like physical hardware, and users can control nearly the entire software stack, from the kernel upward."
  - "At the other extreme of the spectrum are application domain-specific platforms such as Google App Engine ... enforcing an application structure of clean separation between a stateless computation tier and a stateful storage tier"
- The market decision: IaaS won the first battle
  - AWS has the largest share in could market
  - "Google, Microsoft and other cloud companies offered similar interfaces"

IaaS the first model embraced by market

- Early cloud users wanted to recreate the same computing environment in the cloud that they had on the local environment
  - Experienced users; relative low migration cost; low risk if cloud computing could not go far
- Downside
  - Developers needs to manage the VM to set up the environment
  - Needs a lot of administrative knowledge
- The success of cloud gave users more trust on the technology and generated requests for easier management
  - Users are willing to give up control for simpler operation

Providers do more, clients do less



https://www.usenix.org/sites/default/files/conference/protected-files/atc2018_slides_wang.pdf

Function as a Service

- "Serverless computing originated as a design pattern for handling low duty-cycle workloads, such as processing in response to infrequent changes to files stored on the cloud."
  - Eg: "send images from a phone application to the cloud, which should create thumbnail images and then place them on the web"
    - Simple code, but may need an entire web application stack to run it.
    - Running a server for infrequent requests is not cost-effective
- In 2015, Amazon released "AWS Lambda" service
  - User writes a function, specify simple configuration requirements. When the function is invocked, Amazon would start the necessary environment and run it. The user is charged based on the actual execution time.
- The function should be stateless
  - Does not remember its state

Emergence of Serverless Computing

	Characteristic	AWS Serverless Cloud	AWS Serverful Cloud
PROGRAMMER	When the program is run	On event selected by Cloud user	Continuously until explicitly stopped
	Programming Language	JavaScript, Python, Java, Go, C#, etc.[4]	Any
	Program State	Kept in storage (stateless)	Anywhere (stateful or stateless)
	Maximum Memory Size	0.125 - 3 GiB (Cloud user selects)	0.5 - 1952 GiB (Cloud user selects)
	Maximum Local Storage	0.5 GiB	0 - 3600 GiB (Cloud user selects)
	Maximum Run Time	900 seconds	None
	Minimum Accounting Unit	0.1 seconds	60 seconds
	Price per Accounting Unit	$0.0000002 (assuming 0.125 GiB)	$0.0000867 - $0.4080000
	Operating System & Libraries	Cloud provider selects[5]	Cloud user selects
SYSADMIN	Server Instance	Cloud provider selects	Cloud user selects
	Scaling[6]	Cloud provider responsible	Cloud user responsible
	Deployment	Cloud provider responsible	Cloud user responsible
	Fault Tolerance	Cloud provider responsible	Cloud user responsible
	Monitoring	Cloud provider responsible	Cloud user responsible
	Logging	Cloud provider responsible	Cloud user responsible

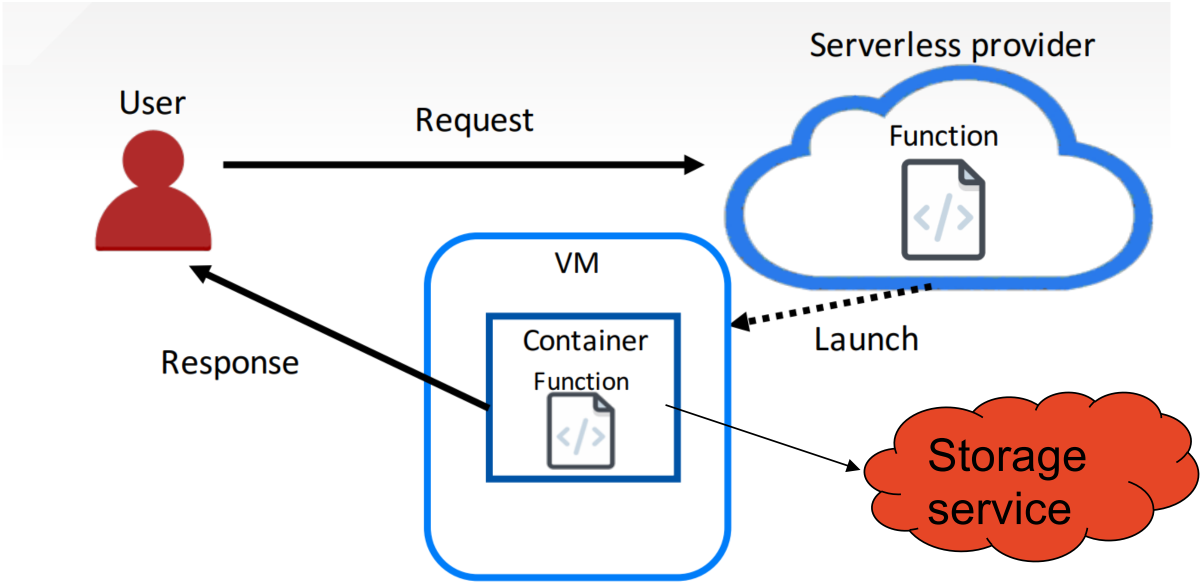
Table 2: Characteristics of serverless cloud functions vs. serverful cloud VMs divided into programming and system administration categories. Specifications and prices correspond to AWS Lambda and to on-demand AWS EC2 instances.

Critical Distinctions

- *Decoupled computation and storage*
    - Computation and Storage are scaled independently, computation is stateless, state is saved elsewhere (e.g. S3, Database, etc)
- *Executing code without managing resource allocation*
    - user provides a piece of code and the cloud automatically provisions resources to execute that code.
- *Paying in proportion to resources used instead of for resources allocated*

How serverless works

- A function usually runs in a container or other type of sandbox with limited resources launched by the provider.



https://www.usenix.org/sites/default/files/conference/protected-files/atc2018_slides_wang.pdf

Attractiveness Of Serverless

- Providers perspective
  - Business growth by bringing new customers and helping existing customers use more service types
  - More predictable resource usages means better utilization.
  - Cloud providers can also utilize less popular computers
- Customer perspective
  - Programming productivity
  - Cost saving

Limitations caused by Storage

- Similar to container, most serverless functions do not have persistent storage.
  - Serverless relies on cloud storage services for maintaining states
  - Certain applications have fine-grained state sharing needs may suffer
    - Latency caused by retrieving and storing states
    - Cost caused by frequent querying
- Serverless functions typically use
  - Object storages like AWS S3, Azure Blob storage, and Google Cloud storage has low storage cost, but high access cost and high access latency
  - Key-value databases, such as AWS DynamoDB, Google Cloud Datastore, or Azure Cosmos DB provide high IOPS, but are expensive and can take a long time to scale up

Amazon Serverless Computing

AWS serverless offerings

Compute
- AWS Lambda and Lambda@Edge
- AWS Fargate

Storage
- Amazon S3
- Amazon EFS

Data Stores
- Amazon DynamoDB
- Amazon Aurora
- Amazon RDS Proxy

API Proxy
- Amazon API Gateway

Application integration
- Amazon SNS
- AWS AppSync
- Amazon SQS
- Amazon EventBridge

Orchestration
- AWS Step Functions

Analytics
- Amazon Kinesis
- Amazon Athena

AWS Lambda

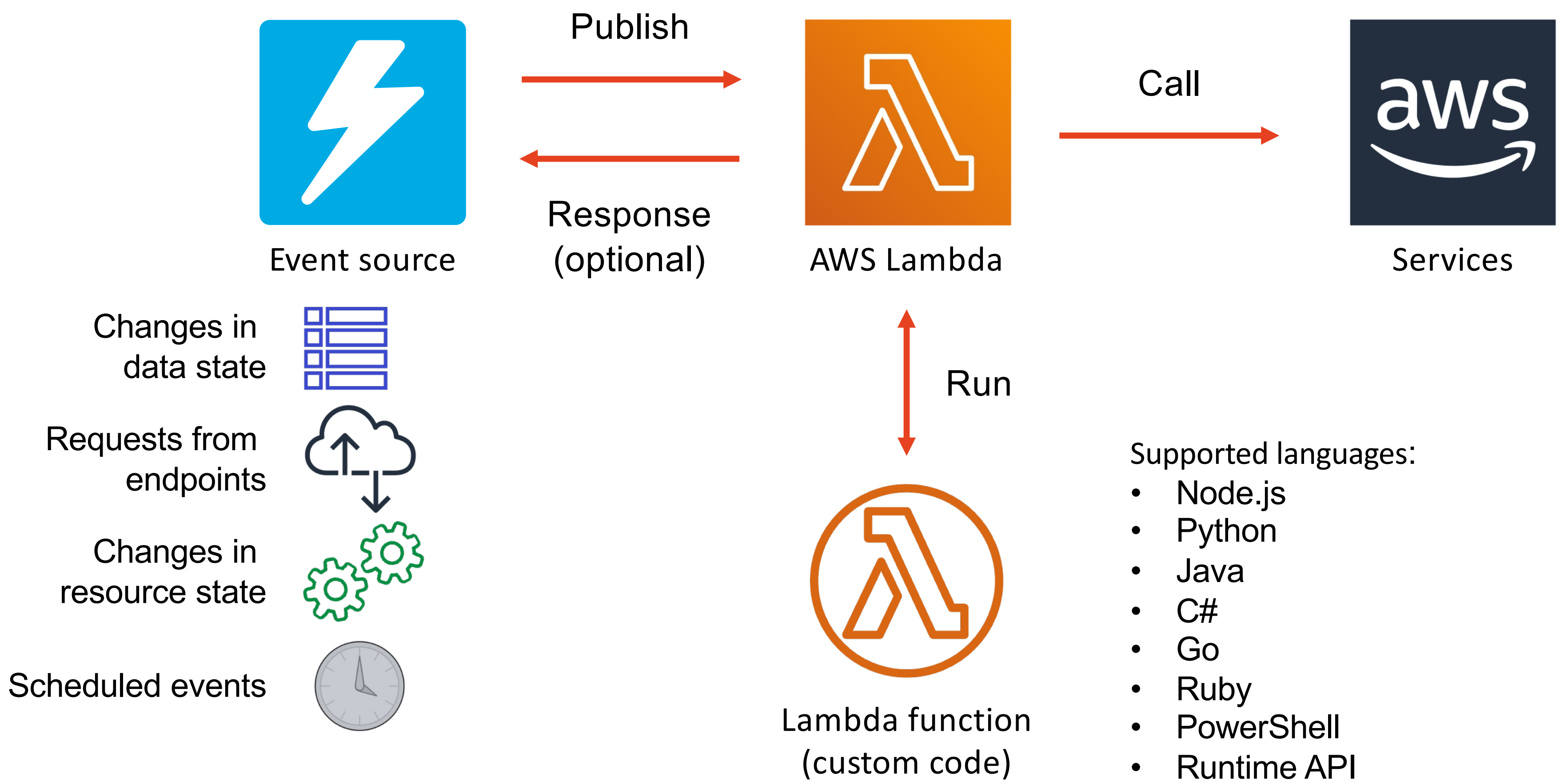
AWS
Lambda

- Is a fully managed compute service
- Runs your code on a schedule or in response to events (for example, changes to an Amazon S3 bucket or an Amazon DynamoDB table)
- Supports Java, Go, PowerShell, Node.js, C#, Python, Ruby, and Runtime API
- Can run at edge locations closer to your users

Basic Concepts

- Function
  - A piece of code that can run by Lambda
- Trigger
  - A resource or configuration that can invoke a Lambda function
- Event
  - A json formatted document that can be passed to Lambda function for processing
  - Many AWS resources can generate events
- Execution environment
  - The container or VM that runs the function
- Runtime
  - The language specific environment in an execution environment
- Deployment package
  - A prebuild package with code and other dependencies
  - Zip file or container image formats are accepted

Typical workflow



Supported languages:
- Node.js
- Python
- Java
- C#
- Go
- Ruby
- PowerShell
- Runtime API

Anatomy of a Lambda function

aws academy

Handler()

Function to be run upon invocation

Event object

Data sent during Lambda function invocation

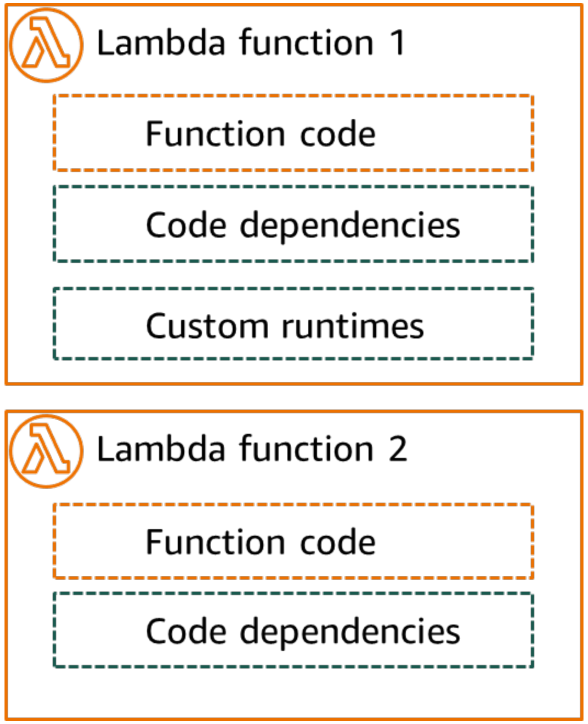
Context object

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

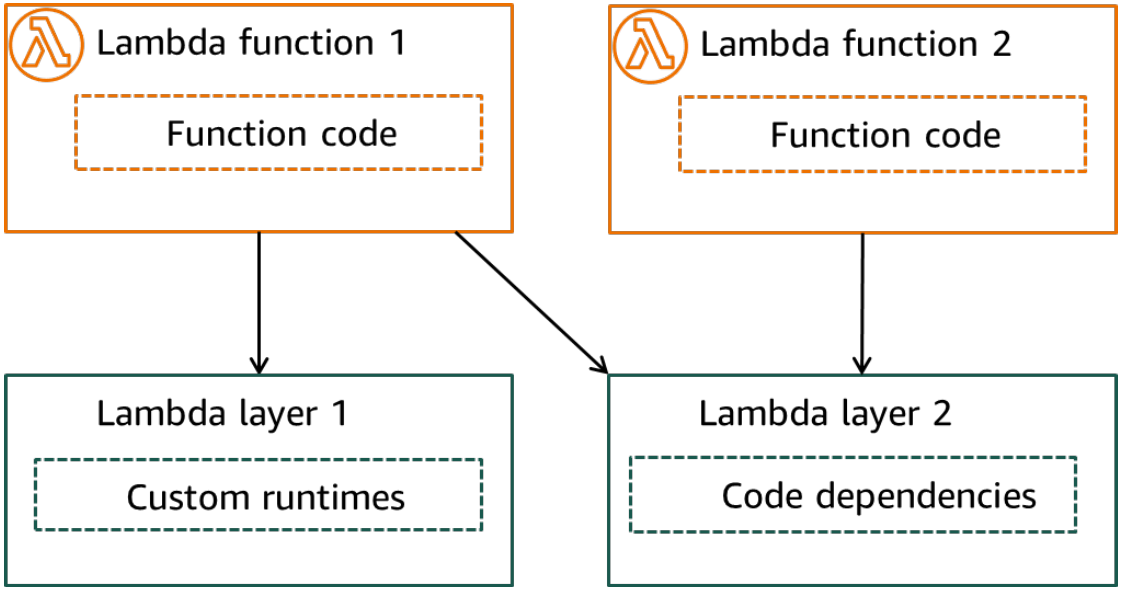
```python
import json

def lambda_handler(event, context):
    # TODO implement
    return {
        'statusCode': 200,
        'body': json.dumps('Hello World')
    }
```

Lambda layers

Without layers



With layers



Lamda local storage

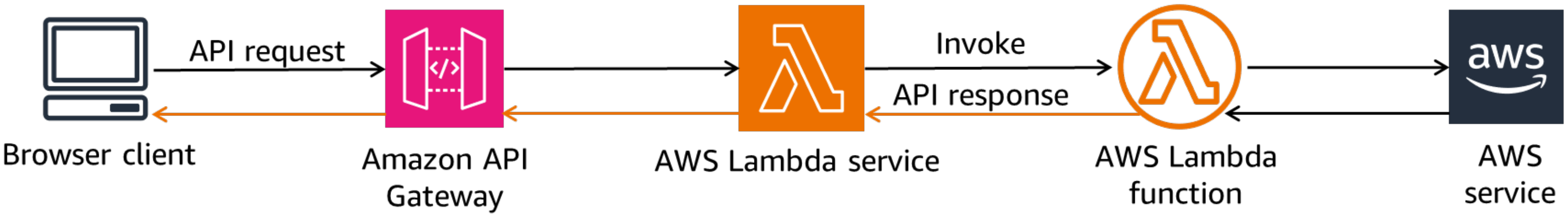
- Temporary local storage /tmp
- The storage is ephemeral, and exists only for the duration of function invocation
- Common use cases:
  - Downloading S2 files for processing
  - Storing intermediate results, etc

Invoking functions

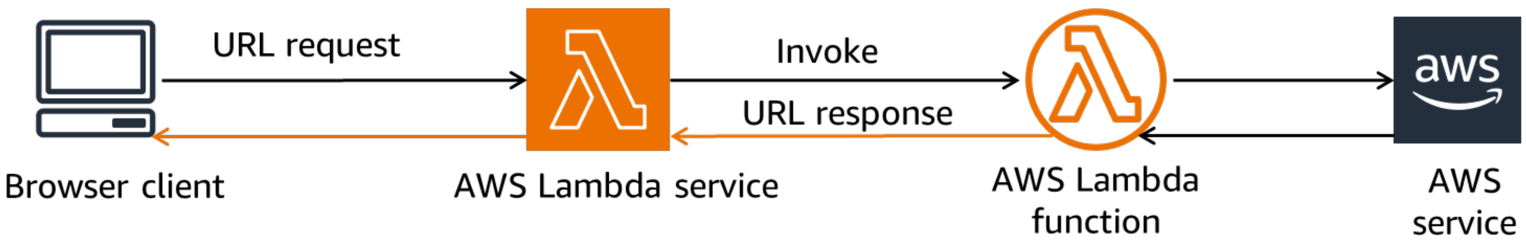
- Through lambda console for testing
- In a program using AWS SDK
- Invoke API
- AWS command line
  - `aws lambda invoke --function-name my-function …`
- **From another service by creating a trigger**

Invoking a synchronous Lambda function

External API request using API Gateway

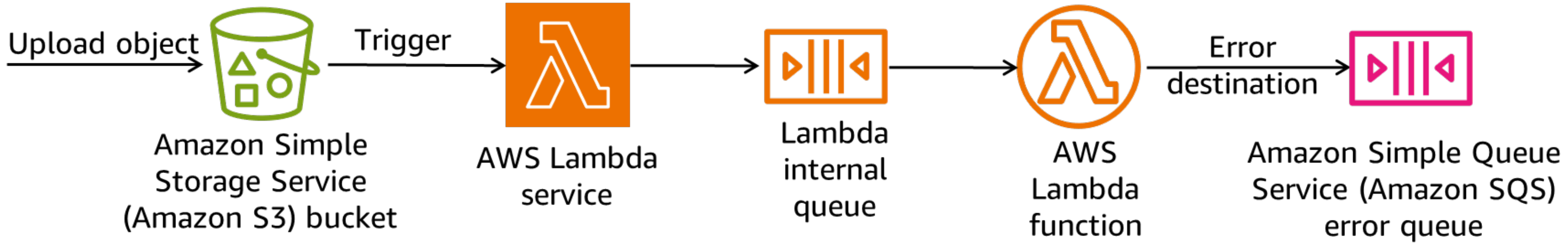


External API request using Lambda function URL

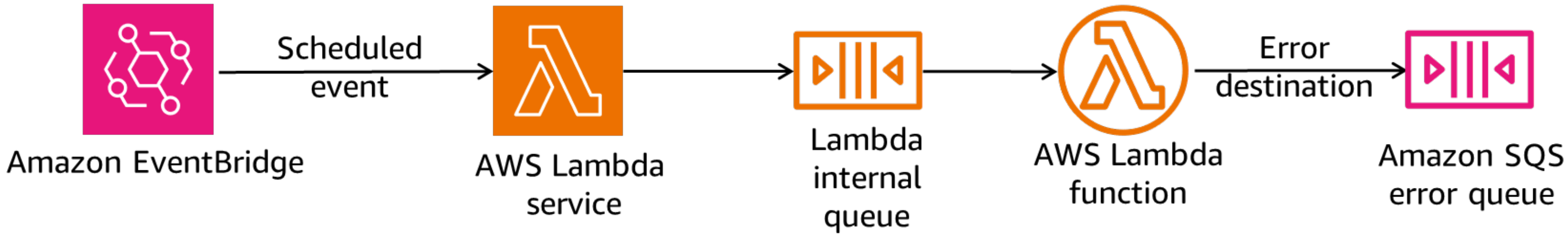


Invoking an asynchronous Lambda function

AWS service trigger event



Scheduled event



Lambda execution environment

Multitenancy and Isolation

- Multitenancy is one of the features of cloud platform
  - The applications from different customers could co-exist on the same hardware
- Multitenancy offers a lot of benefits for both cloud consumers and providers
  - such as elasticity, improved utilization of the servers, etc
- Isolating workloads from each other presents significant challenges to multitenancy
  - Security concern
  - Performance concern

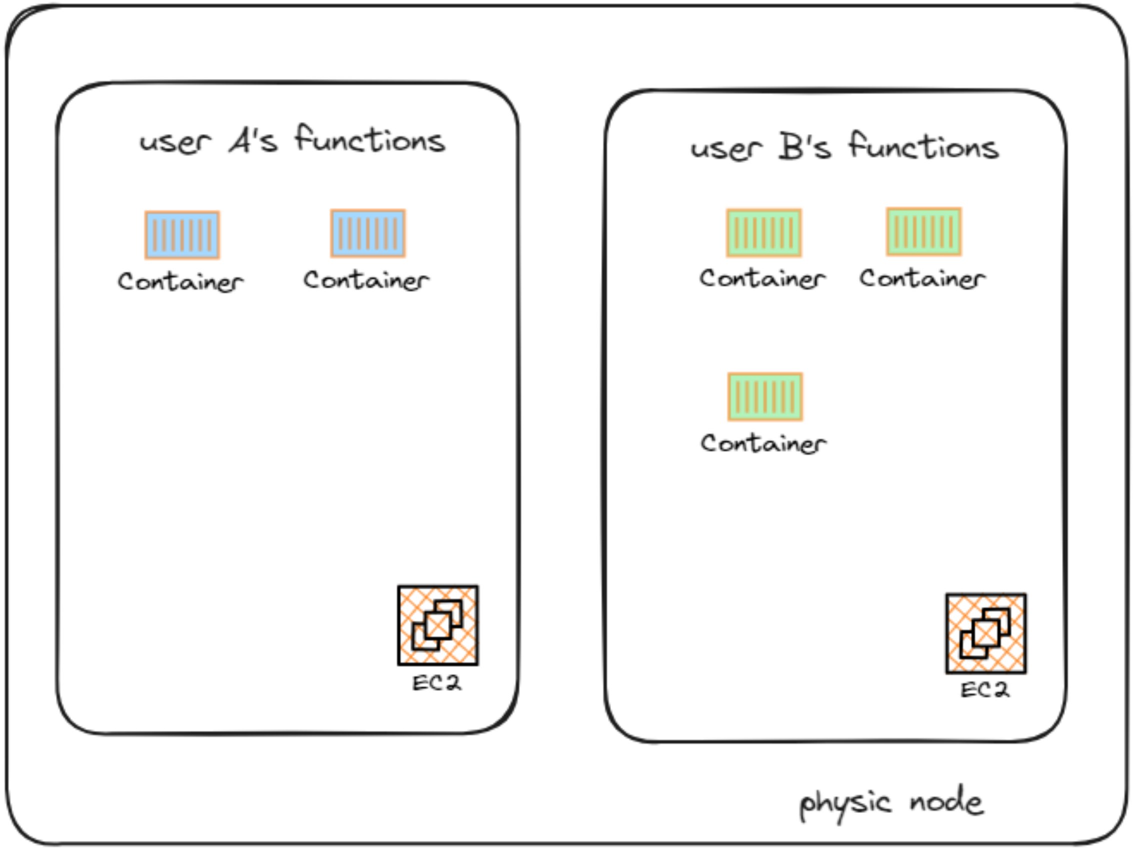
Trusted and untrusted workloads

- Workloads generated by internal applications can trust each other that no one would intentionally harm the system or each other’s environment.
  - The environment does not need to provide the highest level of isolation
  - Cloud focus more on on performance optimization
  - Kubernetes and ACS are designed to run trusted workloads
    - The containers collocate in the same pod give up more isolation for performance gain.
- Workloads generated by different clients cannot have such trust. Some workload could be malicious by design, others could contain vulnerabilities or bugs that could be exploited to compromise the system or data.
  - The environment needs to provide very strong level of isolation

Lambda Execution Environment

- The early version of AWS Lambda uses container technology to isolate functions and virtualization technology to isolate user accounts
  - Each function runs on its own container when invoked.
  - Multiple functions from the same user account may run on the same VM
  - The functions from the different user account will always run on different VMs
- Potential consequences on resource usage:
  - At least one VM for each user account resulted in inefficient resource usage.
  - Scheduling algorithms cannot be optimized for resource usage.

Lambda functions running inside container



MicroVM

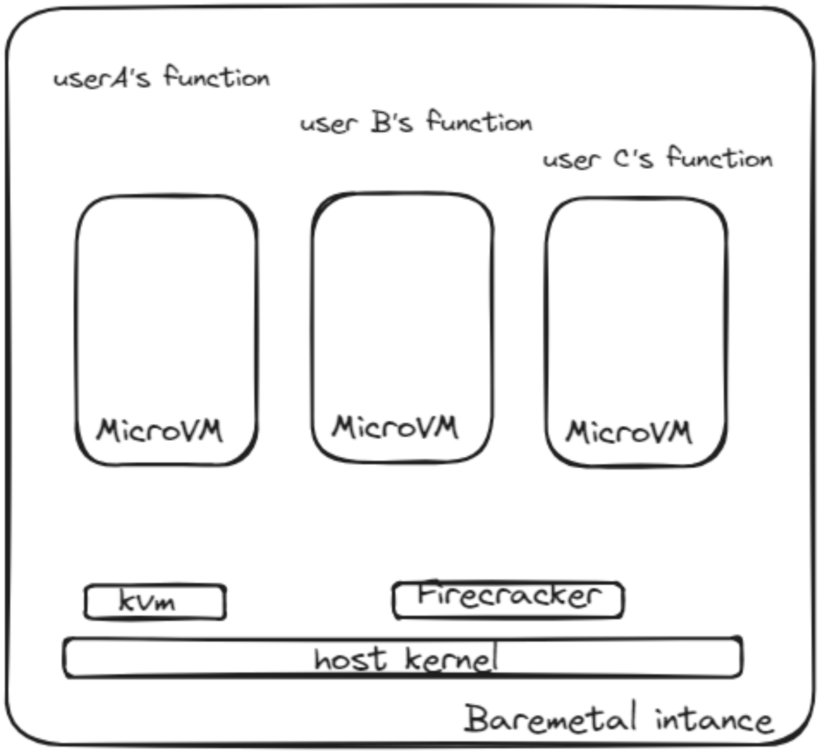
- Serverless computing needs to provide a platform for untrusted workloads to share resources efficiently
  - Workloads from different user accounts can be scheduled on the same node
  - VM level isolation is desirable
    - Each workload runs inside its own VM
- The VM is used to run some function for a short period of time
  - Many traditional OS tools or device drivers are not needed
- A specialized VM, MicroVM, would provide better utilization and performance

MicroVM

- Lambda function now runs in an isolated and secure microVM environment
- MicroVM is a minimum virtual machine that offers high isolation and relatively low resource consumption
  - Somewhere in between container and traditional VM

	Container	MicroVM	Traditional VM
Isolation	Medium	High	High
Resource consumption	Low	Medium	High
Start latency	Low	Medium	High
Use case	Microservices application, deployment everywhere	Serverless computing	General purpose

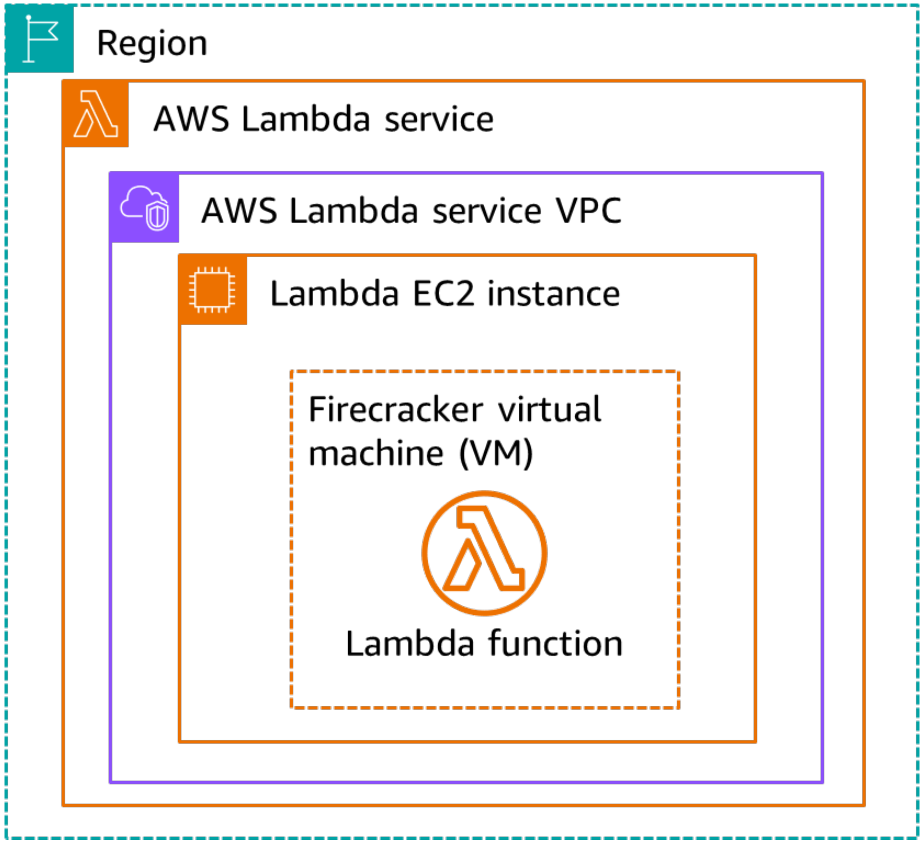
Lambda function supported by Firecracker



Firecracker: lightweight virtualisation

Lambda function location options

In the AWS Lambda service

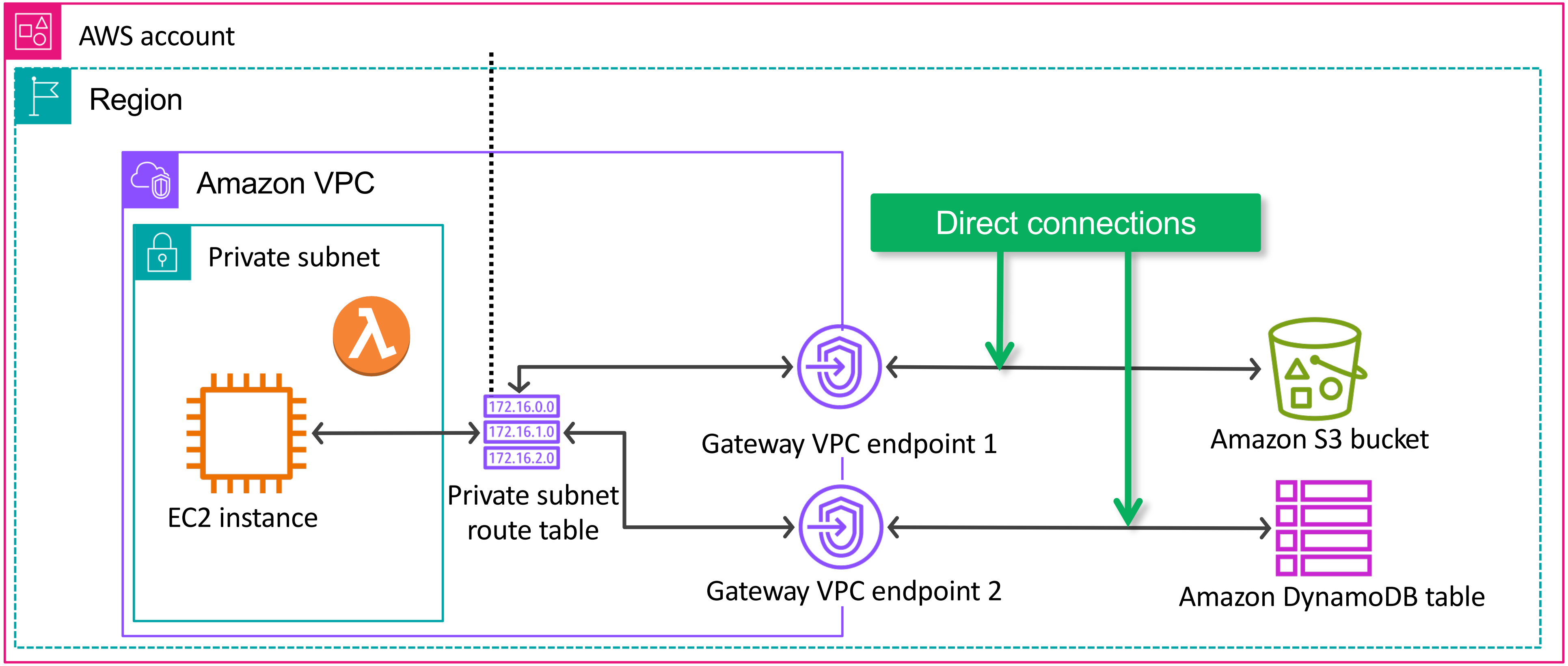


By default:

- Lambda functions run in a Lambda-managed VPC
- It have access to the public internet. This allows them to connect to external services
- We can’t configure or see this VPC
- Lambda functions in the default configuration can access to other outside VPC services like S3, DynamoDB, SecretsManager through preconfigured endpoints
- It cannot access resources on our own VPC, such as RDS

Placing Lambda Function inside a VPC

An easy way is to place the lambda function inside a VPC, on a public or private subnet and configure VPC endpoints for the services not reside in a VPC



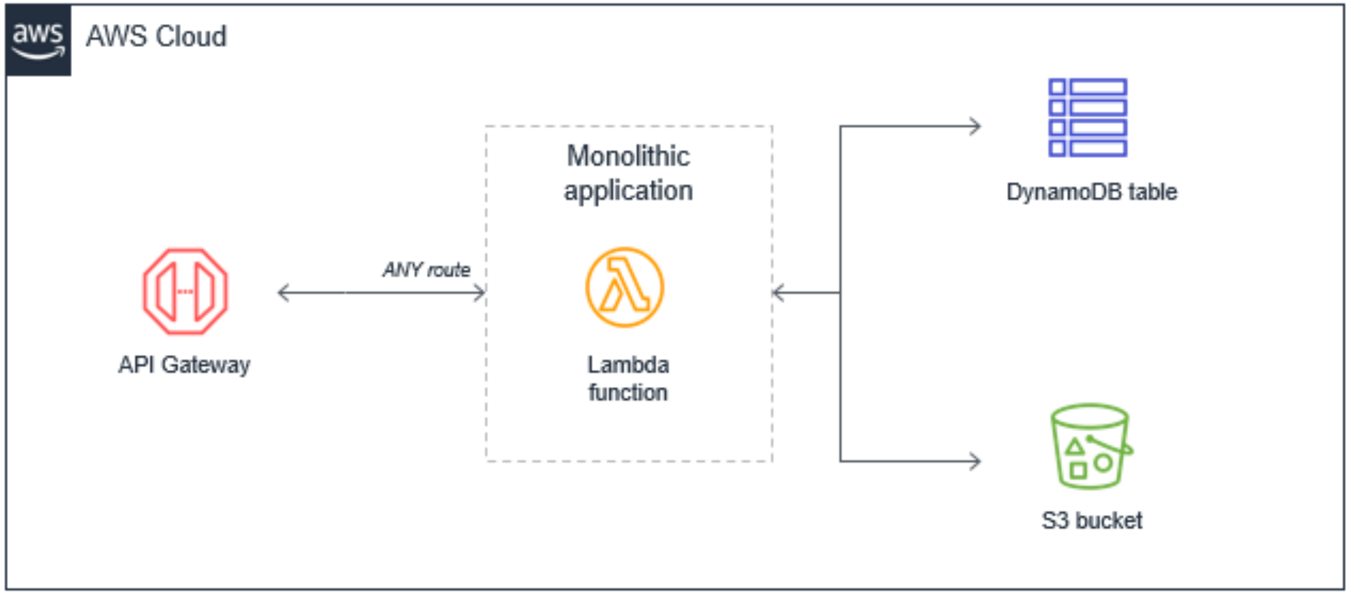
Lambda function anti-patterns

https://docs.aws.amazon.com/lambda/latest/operatorguide/anti-patterns.html

General design principle

- Having many, short functions is preferred over fewer, larger ones
- Each function should be designed to just handle the event passed into the function
- Functions are not expected to have knowledge of the overall workflow or other functions

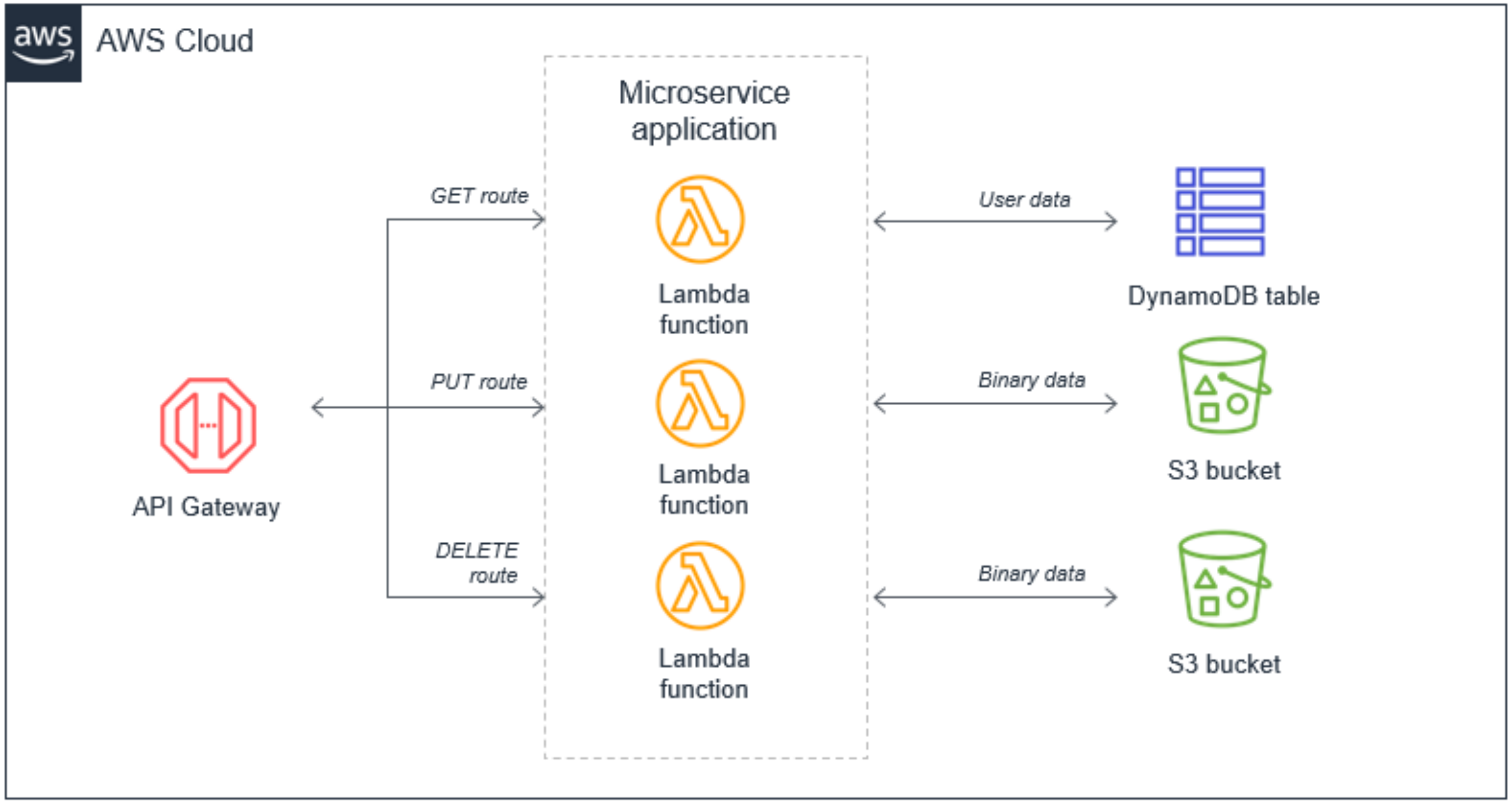
The Lambda monolith



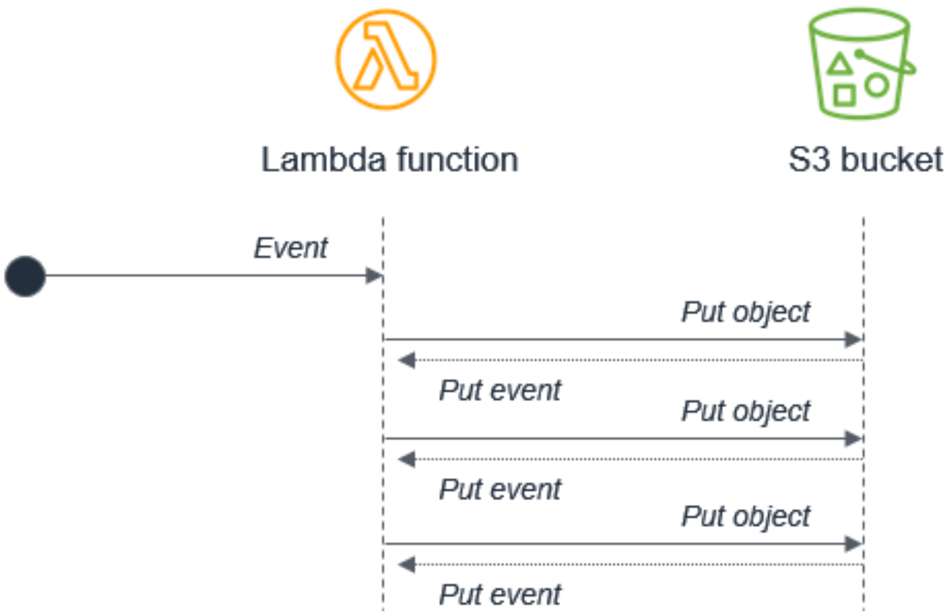
A single function contains all application logic and is triggered for all event;

- Has all the issues of a non-modularized code
  - Hard to upgrade
  - Hard to maintain
  - Hard to reuse
  - Hard to test
- Some cloud specific issues
  - Package size too big
  - Hard to enforce least privilege

Better alternative

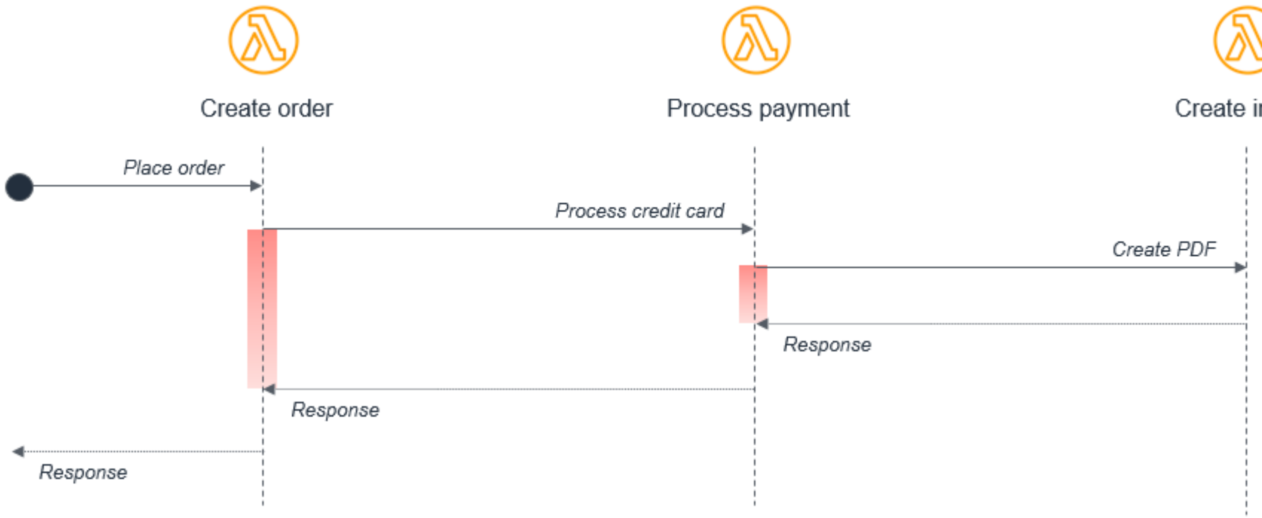


Recursive patterns that cause run-away lambda functions



- Generally, the service or resource that invokes a Lambda function should be different to the service or resource that the function outputs to.
- When an S3 upload event triggers a lambda function which put an object back in the same bucket, trigger event should be carefully configured!

Lambda functions calling Lambda functions

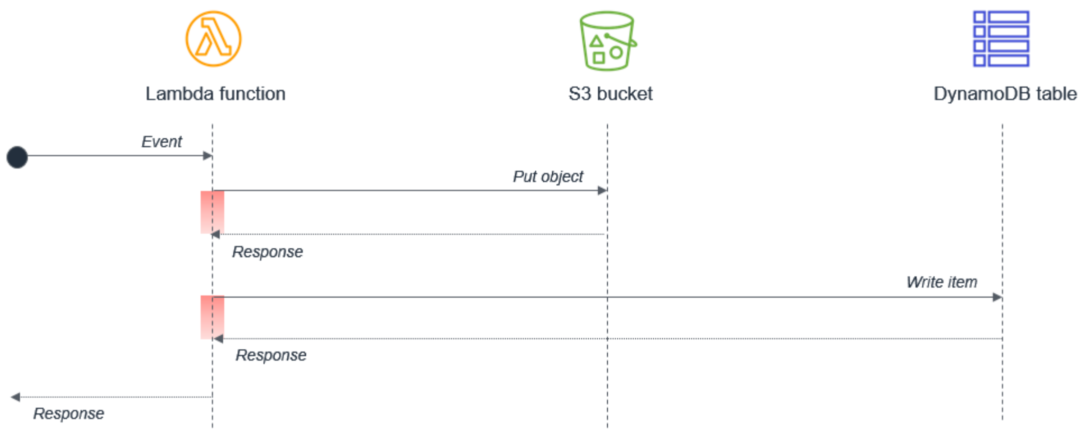


Using Message Queue

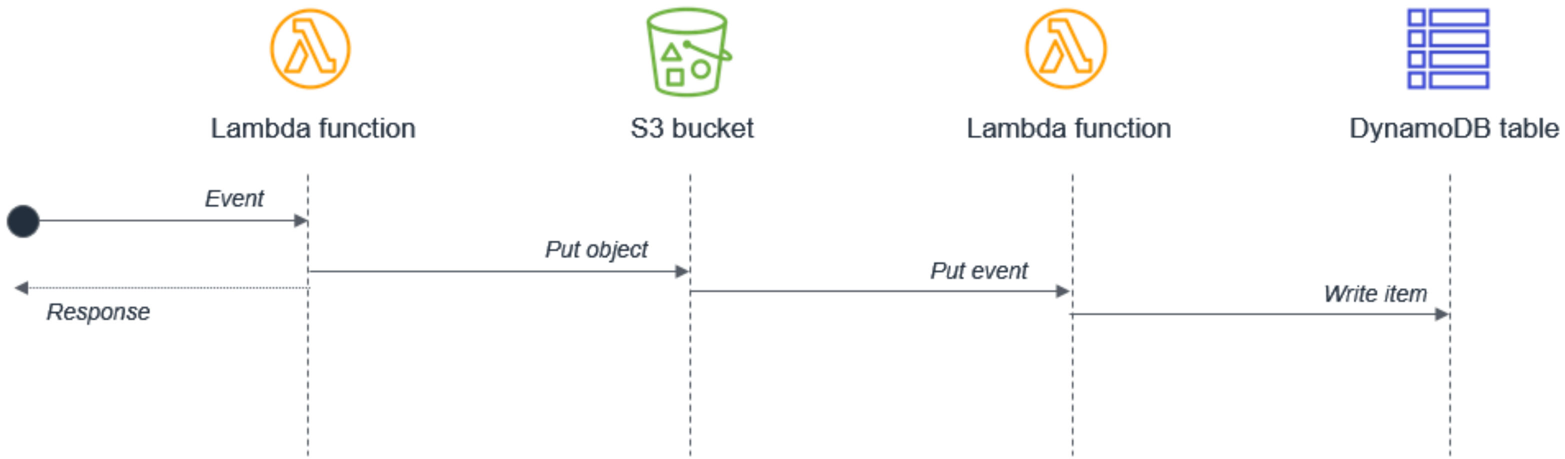
AWS Step Function

- Deep function call stack does not work well in serverless environment
- The above scenario would have three concurrent lambda functions, some are just waiting, which incurs unnecessary cost
- Error handling becomes more complex
- Workflow is limited by the slowest function
- Scaling complexicity

Synchronous waiting within a single lambda function



Using two separate functions



API gateway

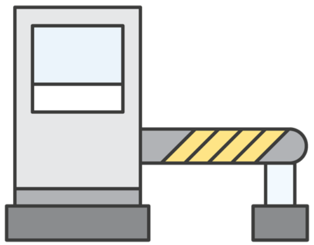
Amazon API Gateway

aws academy

Amazon API Gateway

- Enables you to create, publish, maintain, monitor, and secure APIs that act as entry points to backend resources for your applications
- Handles up to hundreds of thousands of concurrent API calls
- Can handle workloads that run on –
  - Amazon EC2
  - Lambda
  - Any web application
  - Real-time communication applications
- Can host and use multiple versions and stages of your APIs

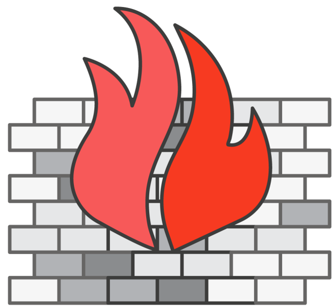
Amazon API Gateway security



Require authorization

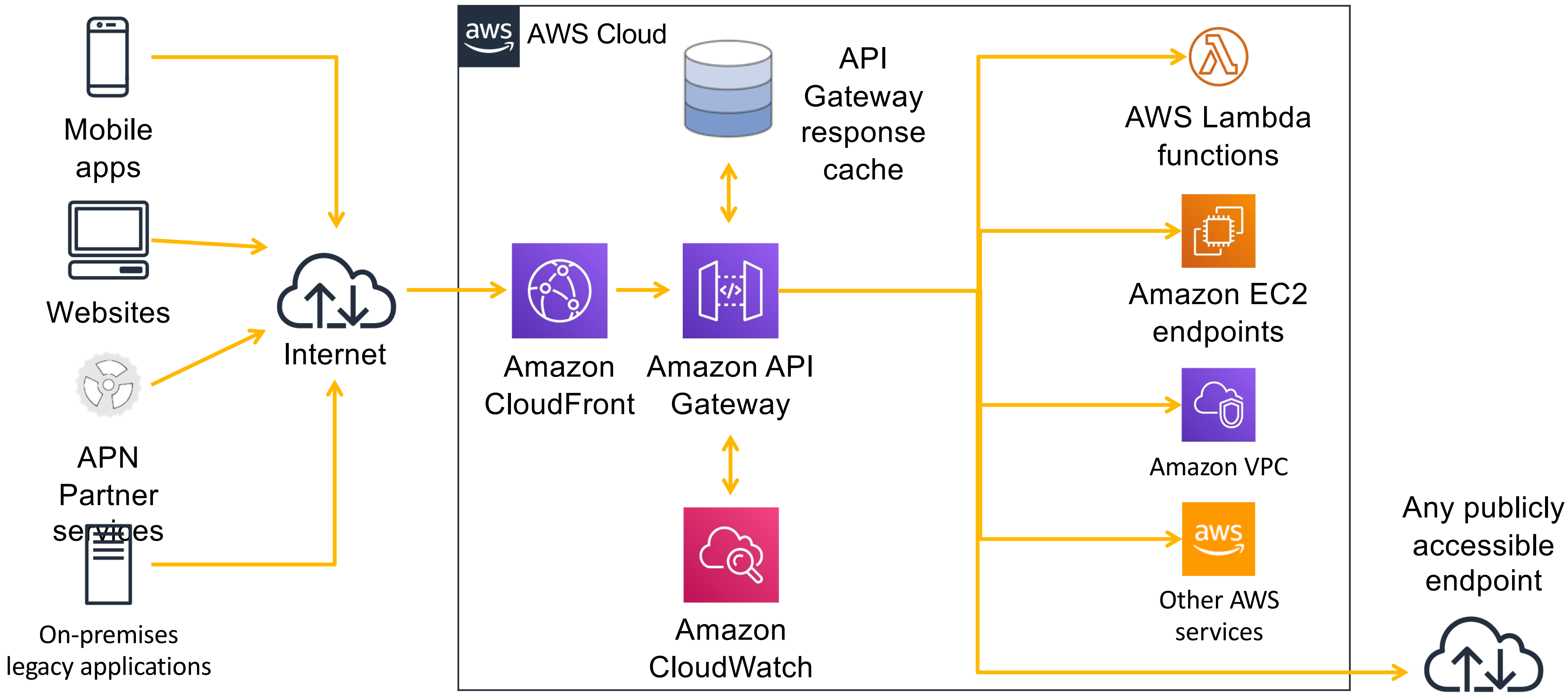
Apply resource policies

Throttling settings

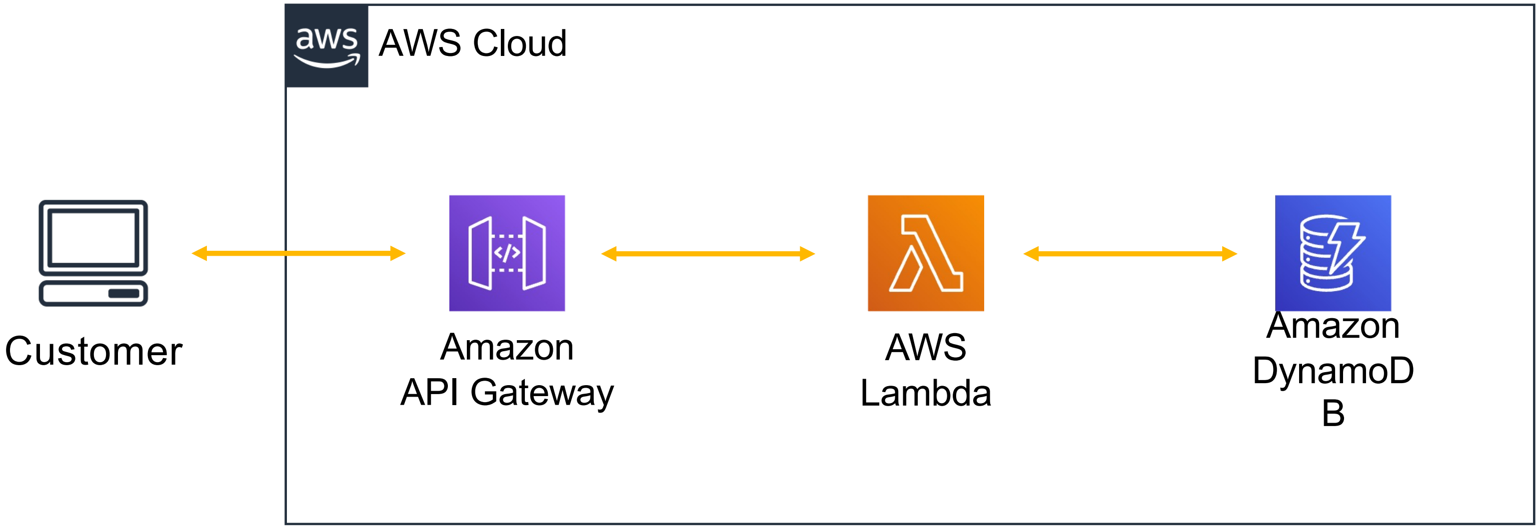


Protection from Distributed Denial of Service (DDoS) and injection attacks

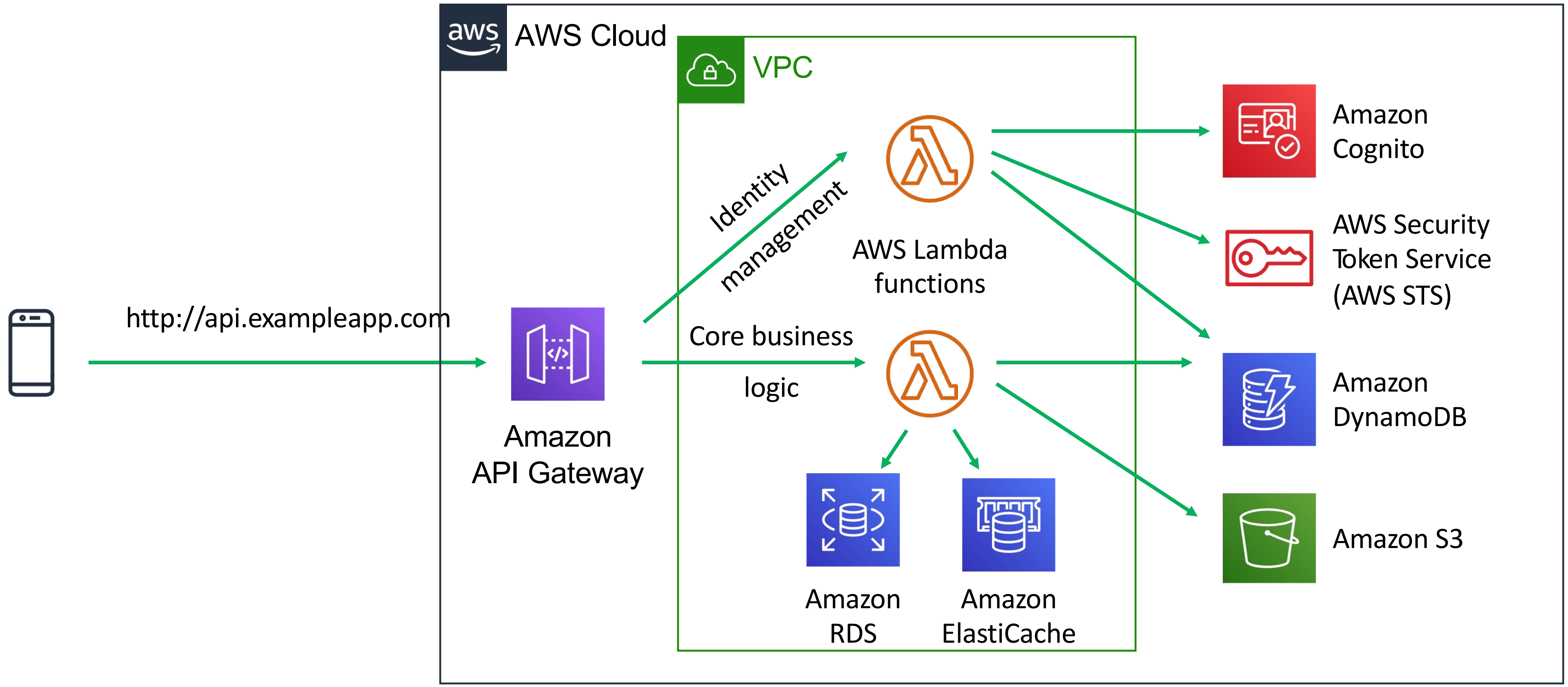
Amazon API Gateway: Common architecture example



Example: RESTful microservices



Example: Serverless mobile backend



Resources

- Jonas, Eric, Johann Schleier-Smith, Vikram Sreekanti, Chia-Che Tsai, Anurag Khandelwal, Qifan Pu, Vaishaal Shankar et al. "Cloud Programming Simplified: A Berkeley View on Serverless Computing." *arXiv preprint arXiv:1902.03383*(2019).
- Wang, Liang, Mengyuan Li, Yinqian Zhang, Thomas Ristenpart, and Michael Swift. "Peeking behind the curtains of serverless platforms." In *2018 USENIX Annual Technical Conference (USENIX}{ATC 18)*, pp. 133-146. 2018.
- Agache, Alexandru, Marc Brooker, Alexandra Iordache, Anthony Liguori, Rolf Neugebauer, Phil Piwonka, and Diana-Maria Popa. "Firecracker: Lightweight Virtualization for Serverless Applications." In *NSDI*, vol. 20, pp. 419-434. 2020.

THE UNIVERSITY OF
SYDNEY