Track 4| Session 2

容器技術和 AWS Lambda 讓您專注「應用優先」

Bob Yeh

Startup Solutions Architect

aws SUMMIT

Agenda

What customers want

Application-first approach

Application-first with AWS Fargate

Application-first with AWS Lambda

Customers have lots of pieces to operate

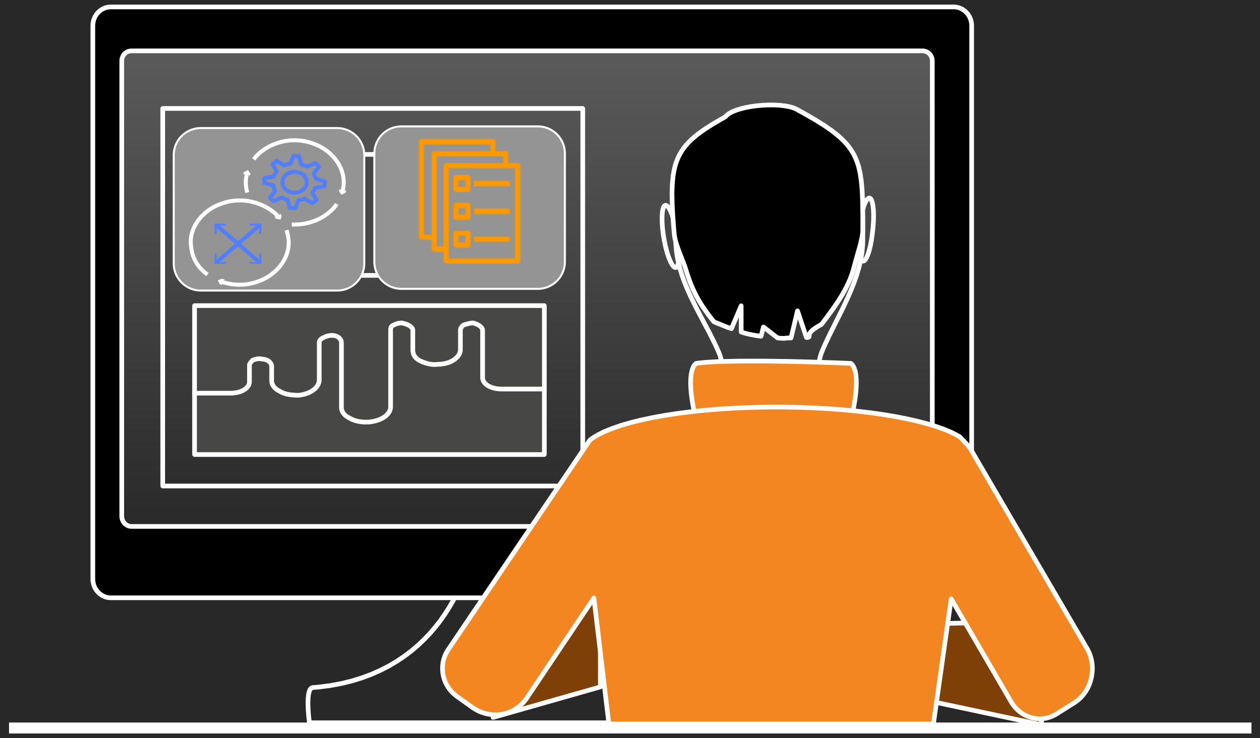
What our customers want

Build applications, not infrastructure

Scale quickly and seamlessly

Security and isolation by design

Application should guide infrastructure



Serverless operations with AWS Fargate and Lambda

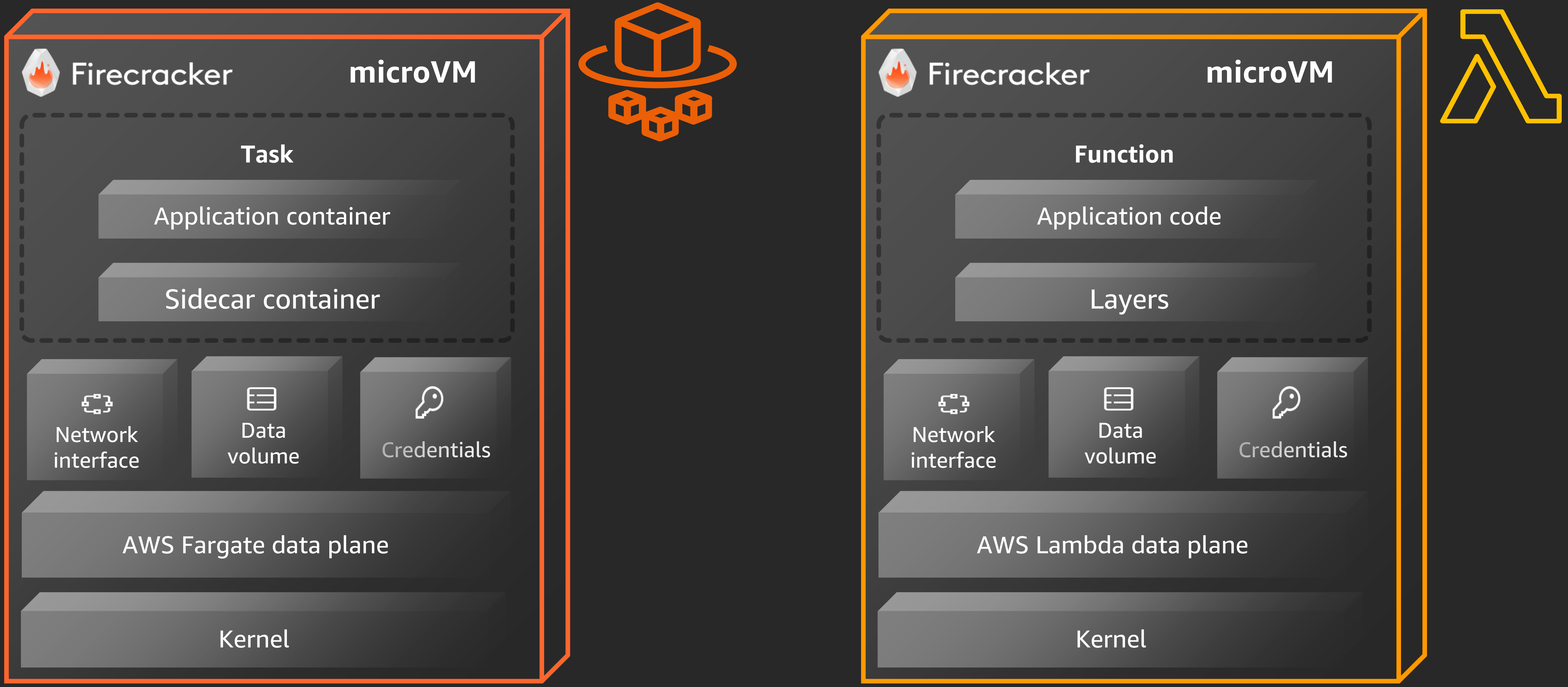
More

Opinionated

Less

	AWS manages	Customer manages	
AWS Lambda Serverless functions	Data source integrations Physical hardware, software, networking, and facilities Provisioning	Application code	
AWS Fargate Serverless containers	Container orchestration, provisioning Cluster scaling Physical hardware, host OS/kernel, networking, and facilities	Application code Data source integrations Security config and updates	Network config Management tasks
Amazon ECS/EKS Container-management -as-a-service	Container orchestration control plane Physical hardware, software, networking, and facilities	Application code Data source integrations Work clusters	Security config and updates, network config, firewall, and management tasks
Amazon EC2 Infrastructure-as-a-service	Physical hardware, software, networking, and facilities	Application code Data source integrations Scaling Security config and updates Network config	Management tasks Provisioning, managing scaling and patching of servers

Execution isolation boundary



Application-first with AWS Fargate

aws SUMMIT

Goal

AWS Fargate

Allow customers to run containers without managing the underlying virtual machines

AWS Fargate design tenets

Security

Ensuring the security of the infrastructure that is underlying customer containers is our primary tenet

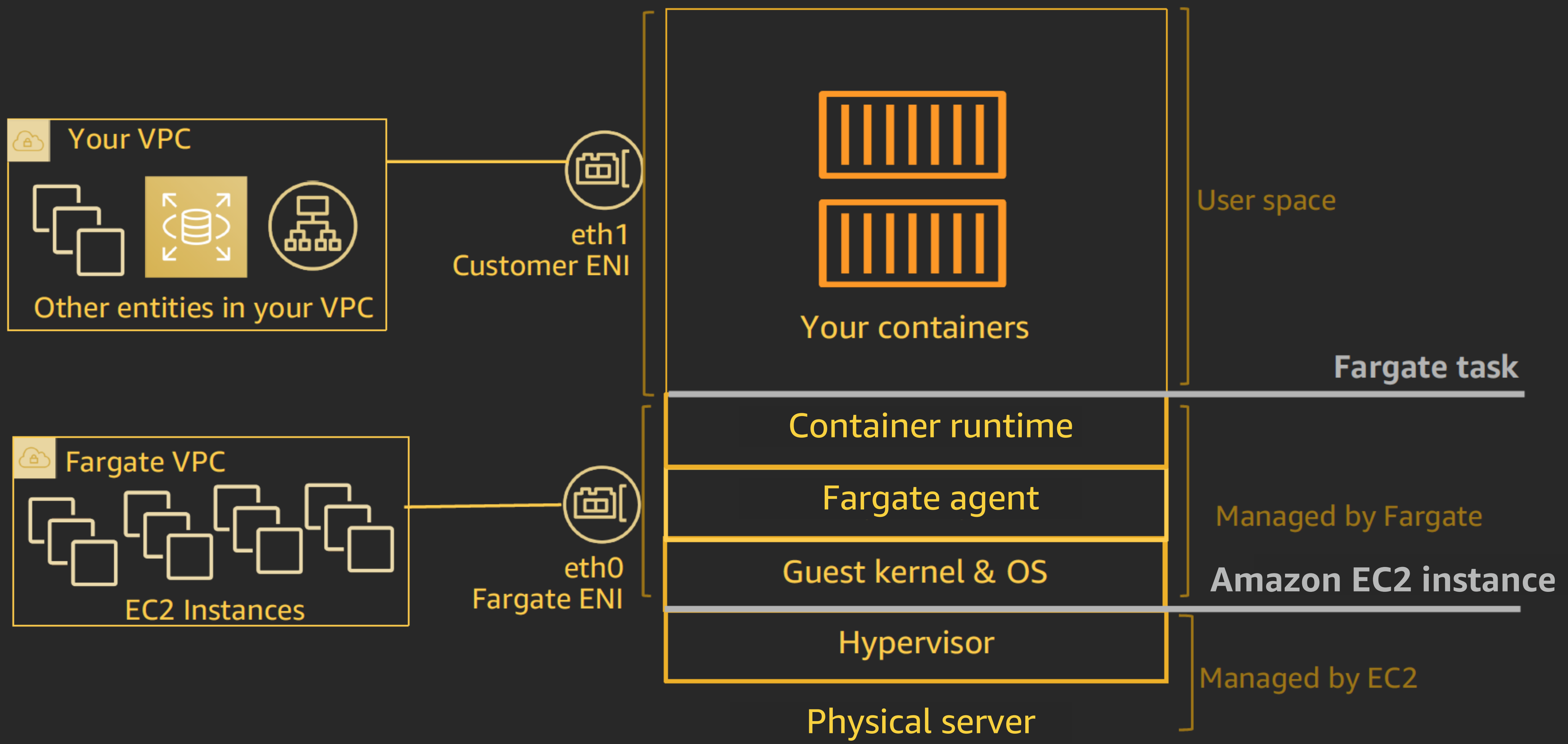
Availability and scalability

This includes maintaining both uptime of running containers and elasticity of the platform to support rapid scale-out and scale-in of containers with high reliability

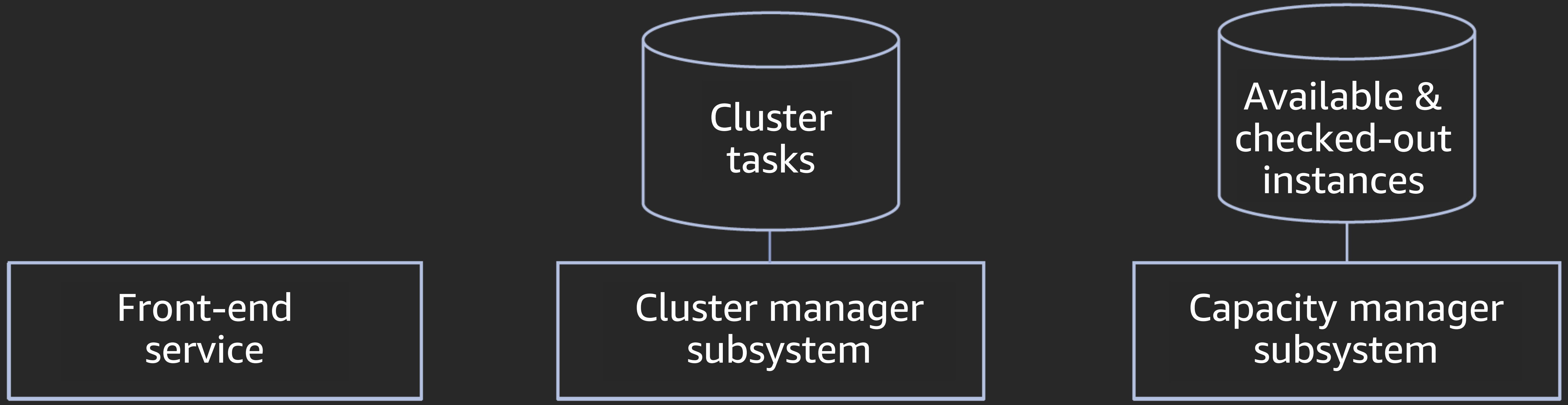
Operational efficiency

Maintain high resources utilization of the underlying fleet to reduce operational costs for the business

AWS Fargate data plane



AWS Fargate control plane



Front-end service

Cluster tasks

Cluster manager subsystem

Available & checked-out instances

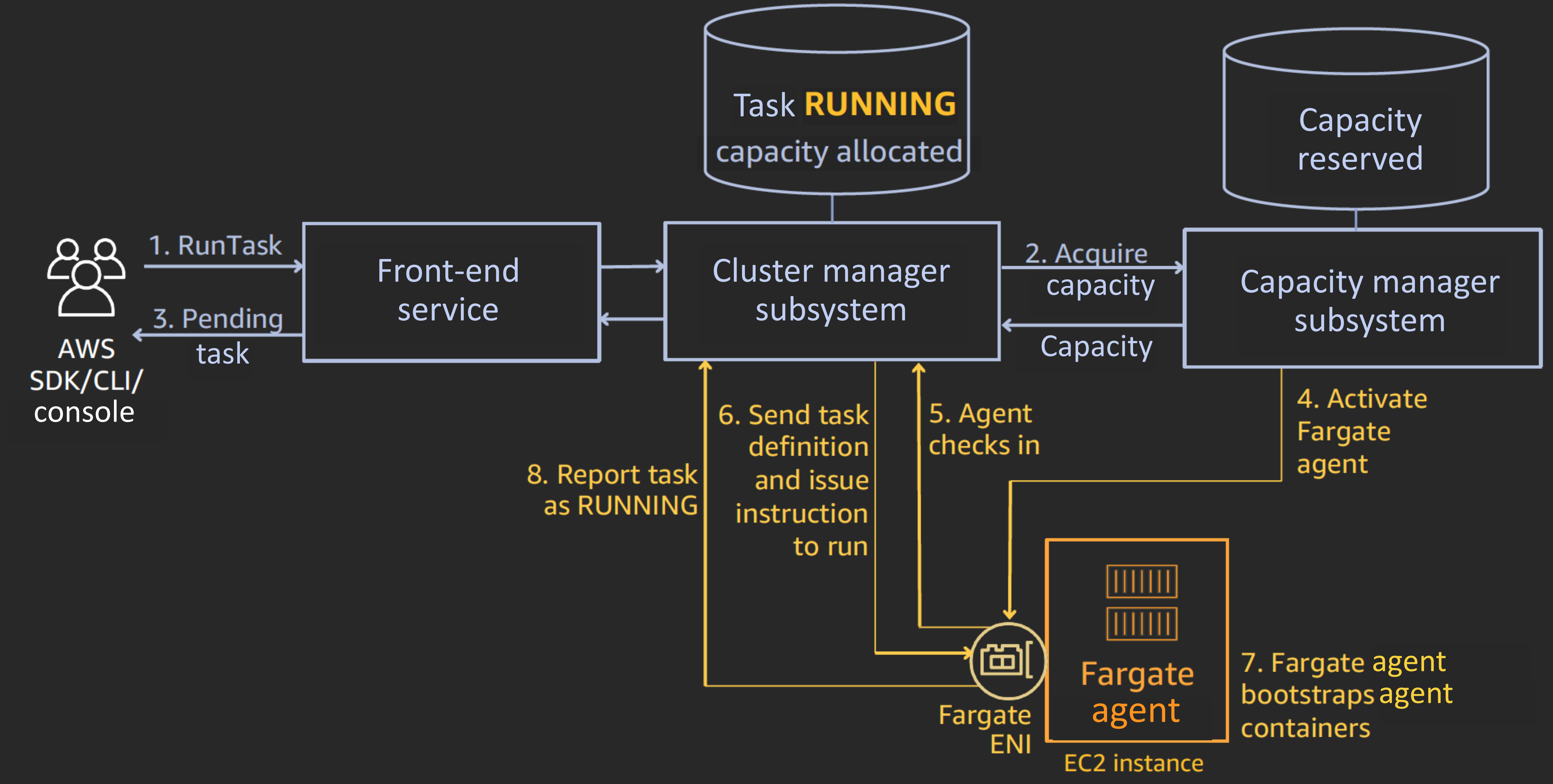
Capacity manager subsystem

- Entry point
- Authentication
- Authorization
- Limit enforcement

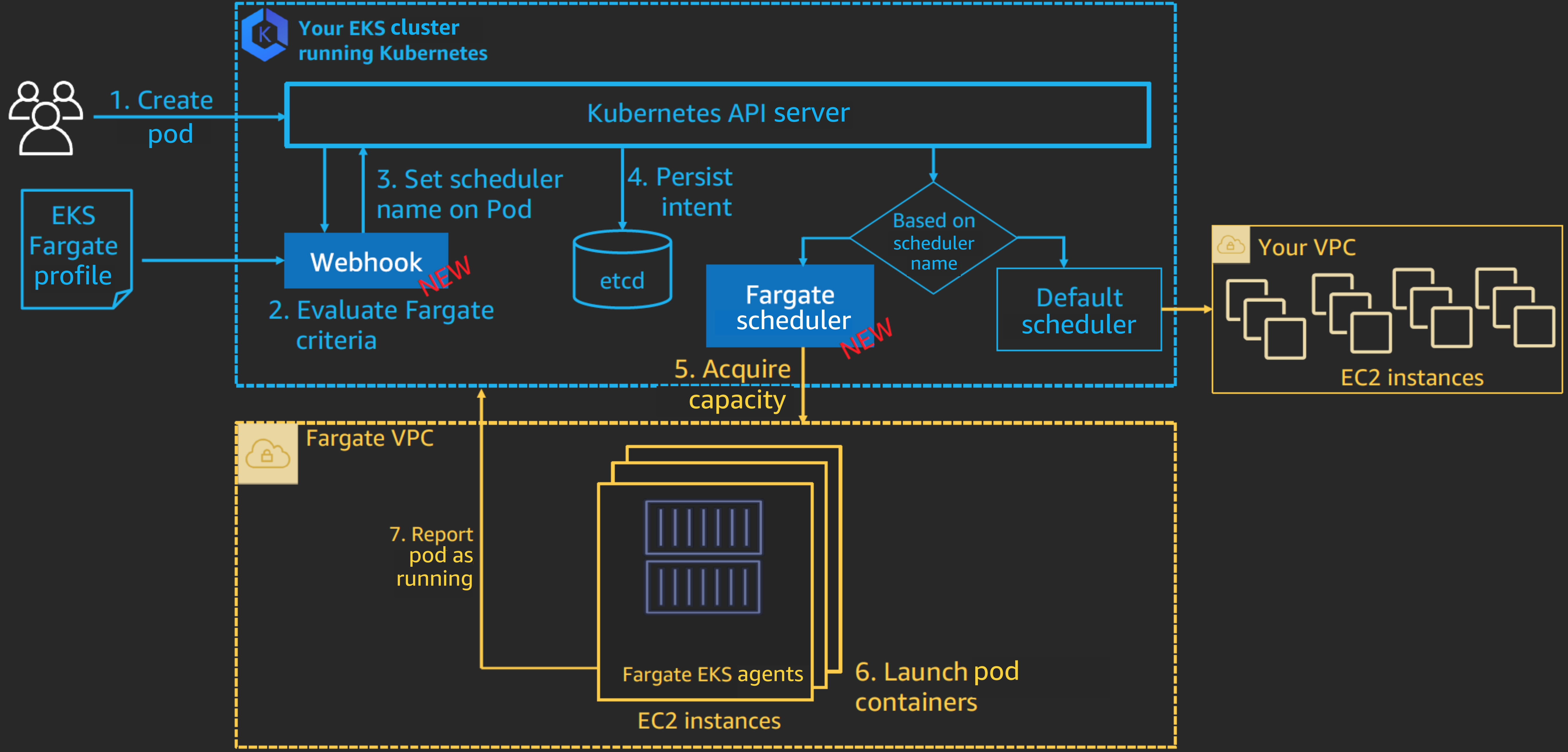
- Keeps state about clusters and tasks
- Communicates with the data plane and drives task state changes

- Keeps state about instances
- Placement of tasks on instances
- Replenishes capacity

RunTask call flow



Amazon EKS on AWS Fargate architecture



AWS Fargate VPC integration

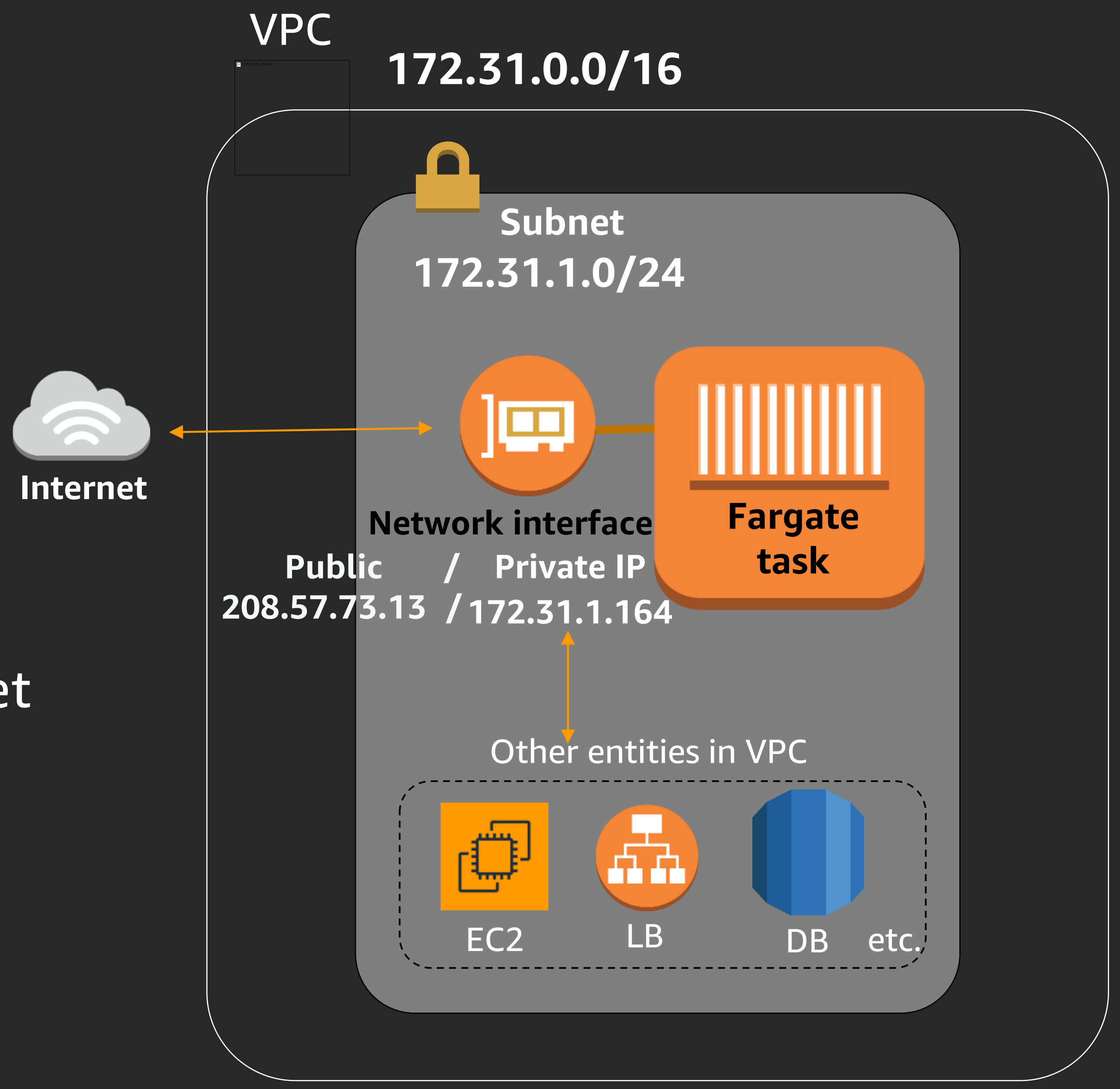
Launch your Fargate tasks into subnets

Under the hood

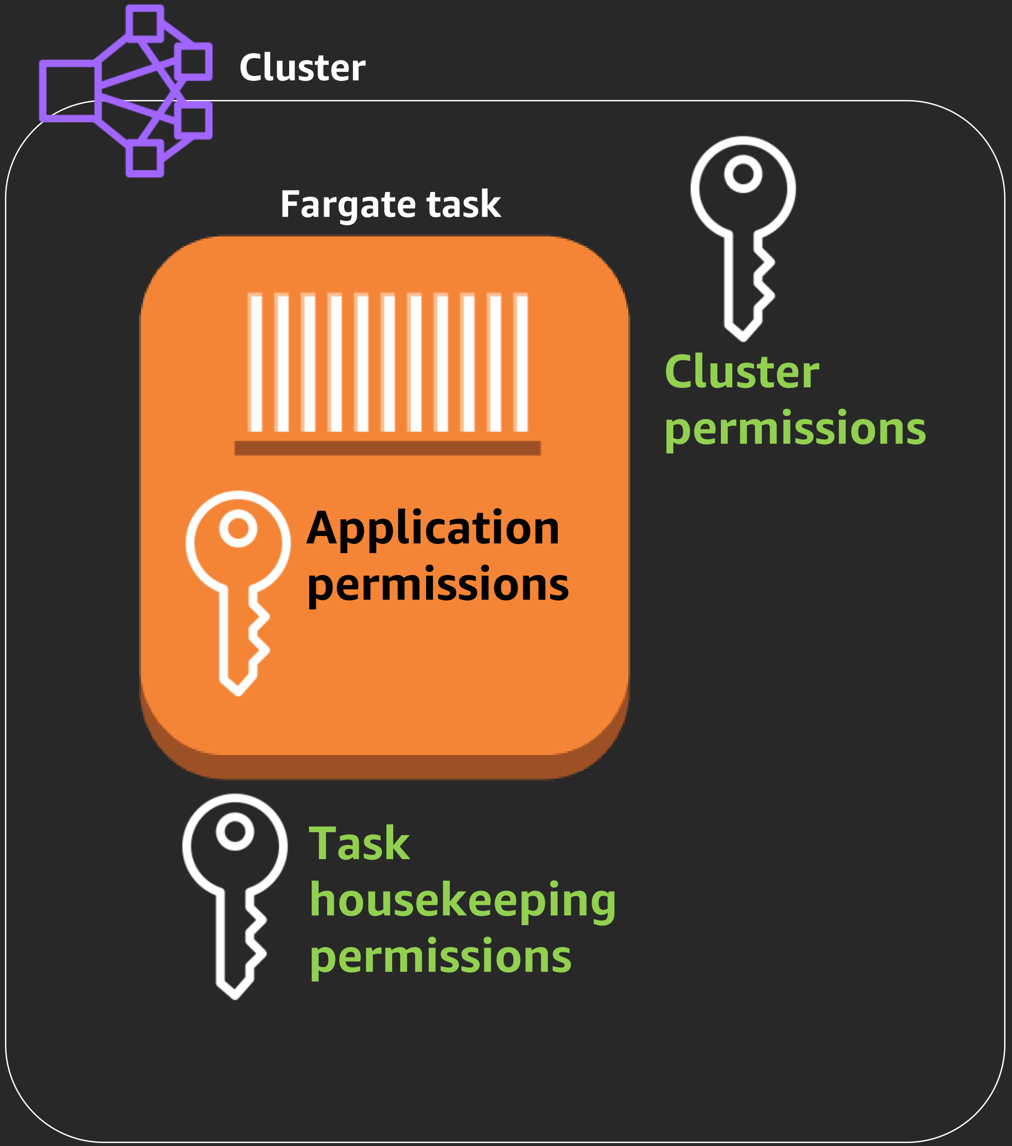
- We create an elastic network interface
- The network interface is allocated a private IP from your subnet
- The network interface is attached to your task
- Your task now has a private IP from your subnet

You can assign public IPs to your tasks

Configure security groups to control inbound and outbound traffic



AWS Fargate IAM permission types



Cluster permissions

Control who can launch/describe tasks in your cluster

Application permissions

Allows your application containers to access AWS resources securely

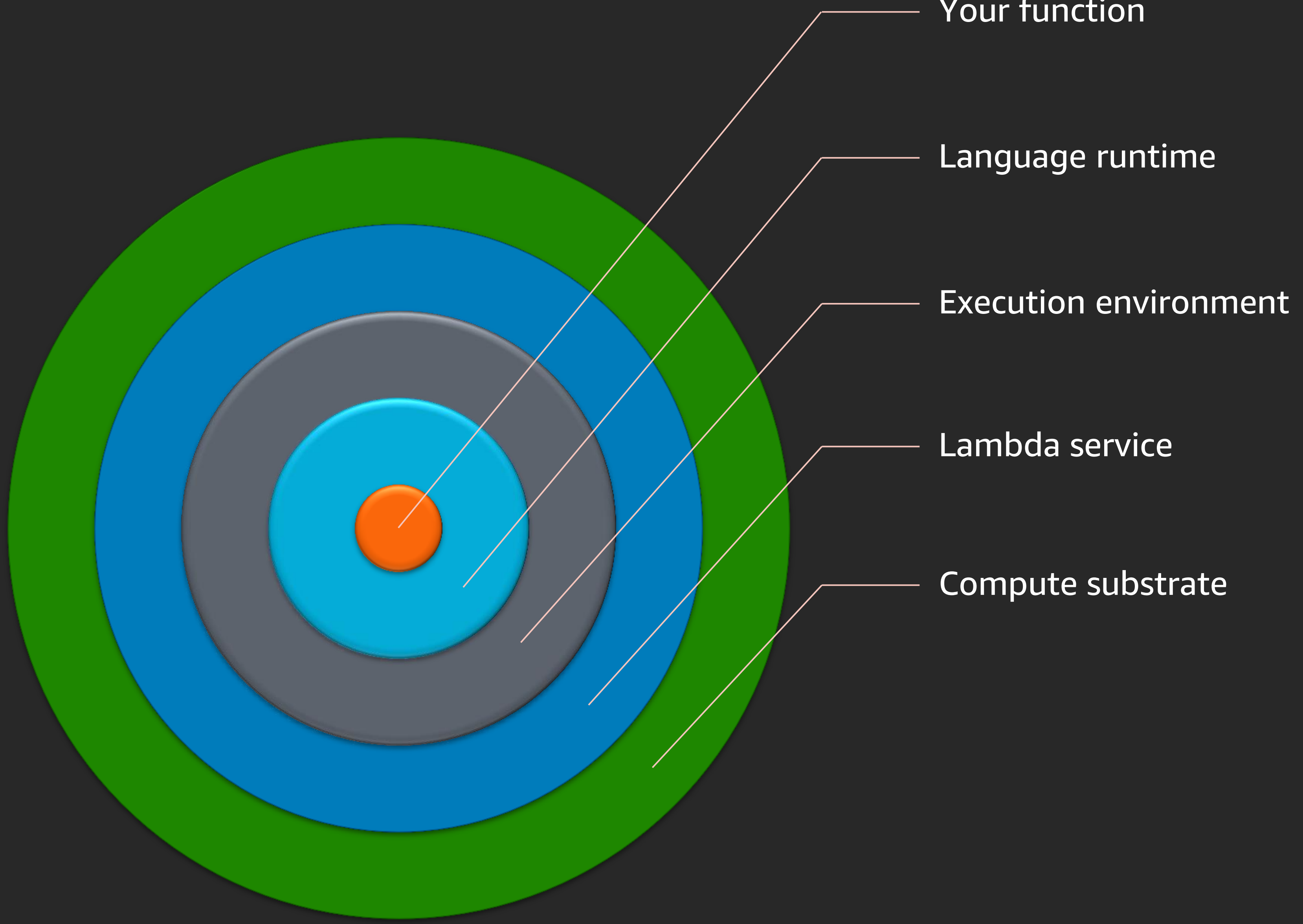
Housekeeping permissions

Allows us to perform housekeeping activities around your task

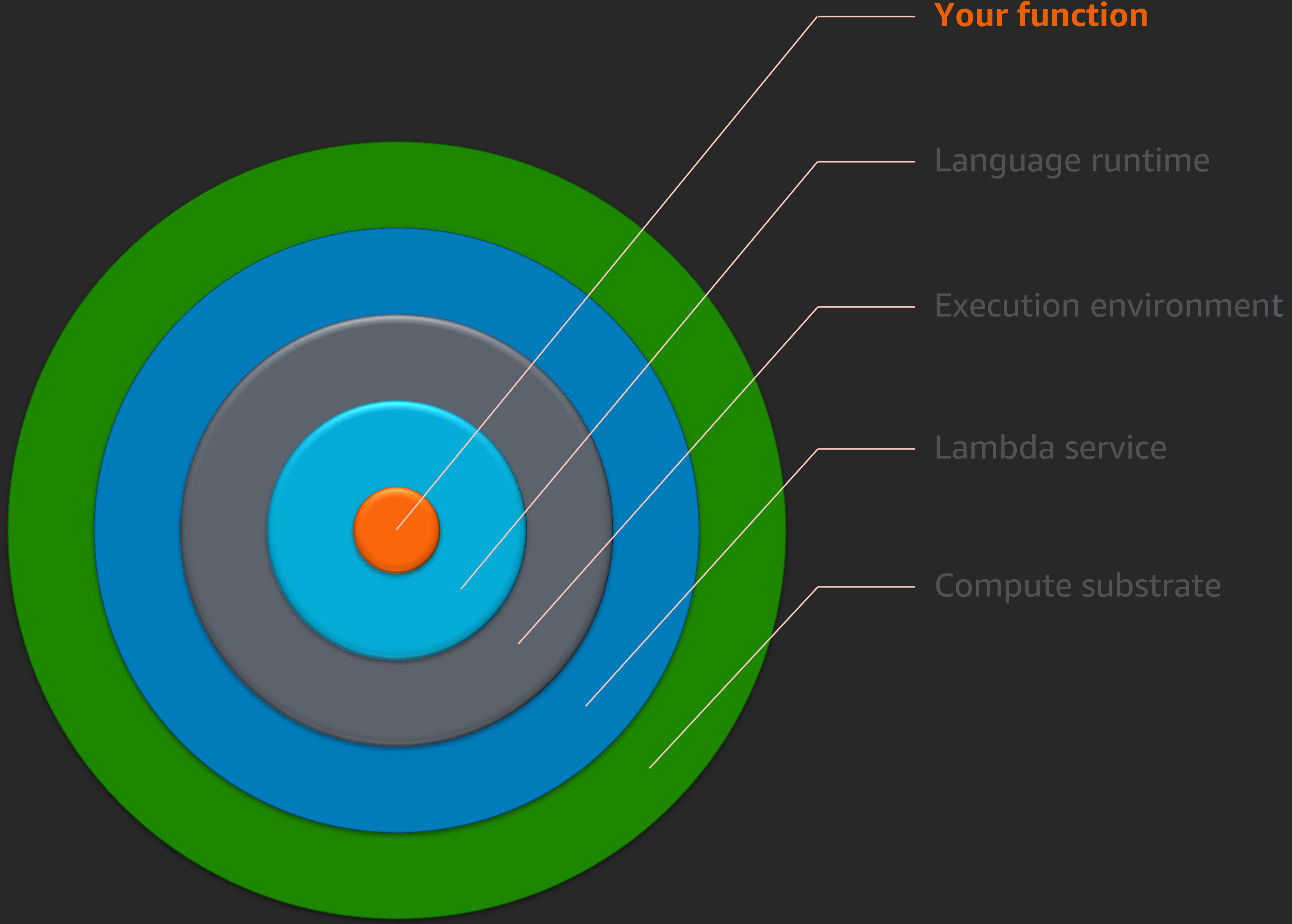
- Amazon ECR image pull
- Amazon CloudWatch Logs pushing
- Network interface creation
- Register/deregister targets into ELB

Application-first with AWS Lambda

Anatomy of an AWS Lambda function



Anatomy of an AWS Lambda function



Anatomy of an AWS 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)

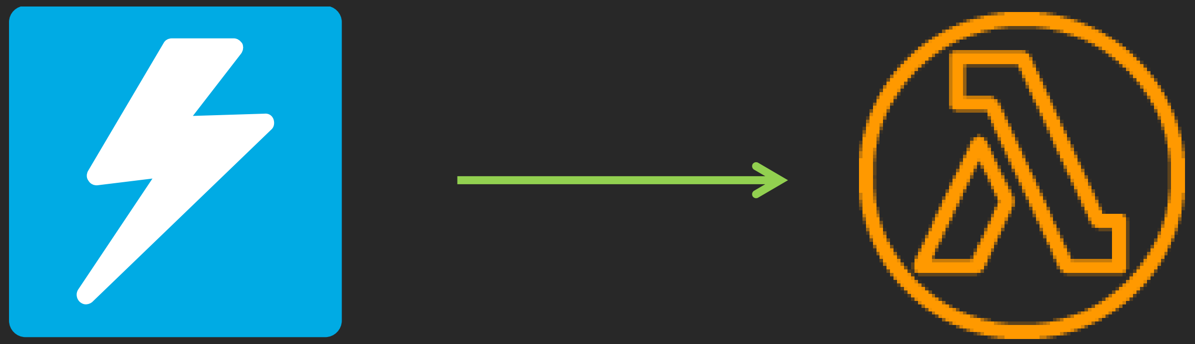
```
import json

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

Serverless applications

Event source

Function



Changes in data state

Requests to endpoints

Changes in resource state

Node.js
Python
Java
C#
Go
Ruby
Runtime API

Anatomy of an AWS Lambda function

```
Function myhandler(event, context) {
    <Event handling logic> {
             result = SubfunctionA()
         }else {
                 result = SubfunctionB()

    return result;
}
```

Your handler

```
Import sdk
Import http-lib
Import ham-sandwich

Pre-handler-secret-getter()
Pre-handler-db-connect()

Function myhandler(event, context) {
    <Event handling logic> {
             result = SubfunctionA()
         }else {
                 result = SubfunctionB()

    return result;
}
```

Your handler

```
Import sdk
Import http-lib
Import ham-sandwich

Pre-handler-secret-getter()
Pre-handler-db-connect()

Function myhandler(event, context) {
    <Event handling logic> {
            result = SubfunctionA()
        }else {
                result = SubfunctionB()

    return result;
}
```

Dependencies, configuration information, common helper functions

Your handler

Pre-handler code, dependencies, variables

- Import only what you need
  - Where possible, trim down SDKs and other libraries to the specific bits required
- Pre-handler code is great for establishing connections, but be prepared to then handle reconnections in further executions
- **Remember** – execution environments are reused
  - Lazily load variables in the global scope
  - Don't load it if you don't need it – cold starts are affected
  - Clear out used variables so you don't run into leftover state

```
Import sdk
Import http-lib
Import ham-sandwich

Pre-handler-secret-getter()
Pre-handler-db-connect()

Function myhandler(event,
context) {
....
```

```
Import sdk
Import http-lib
Import ham-sandwich

Pre-handler-secret-getter()
Pre-handler-db-connect()
```

Dependencies, configuration information, common helper functions

```
Function myhandler(event, context) {
    <Event handling logic> {
            result = SubfunctionA()
        }else {
                result = SubfunctionB()

    return result;
}
```

Your handler

```
Function Pre-handler-secret-getter() {
}

Function Pre-handler-db-connect(){
}
```

```
Import sdk
Import http-lib
Import ham-sandwich

Pre-handler-secret-getter()
Pre-handler-db-connect()
```

Dependencies, configuration information, common helper functions

```
Function myhandler(event, context) {
    <Event handling logic> {
            result = SubfunctionA()
        }else {
                result = SubfunctionB()

    return result;
}
```

Your handler

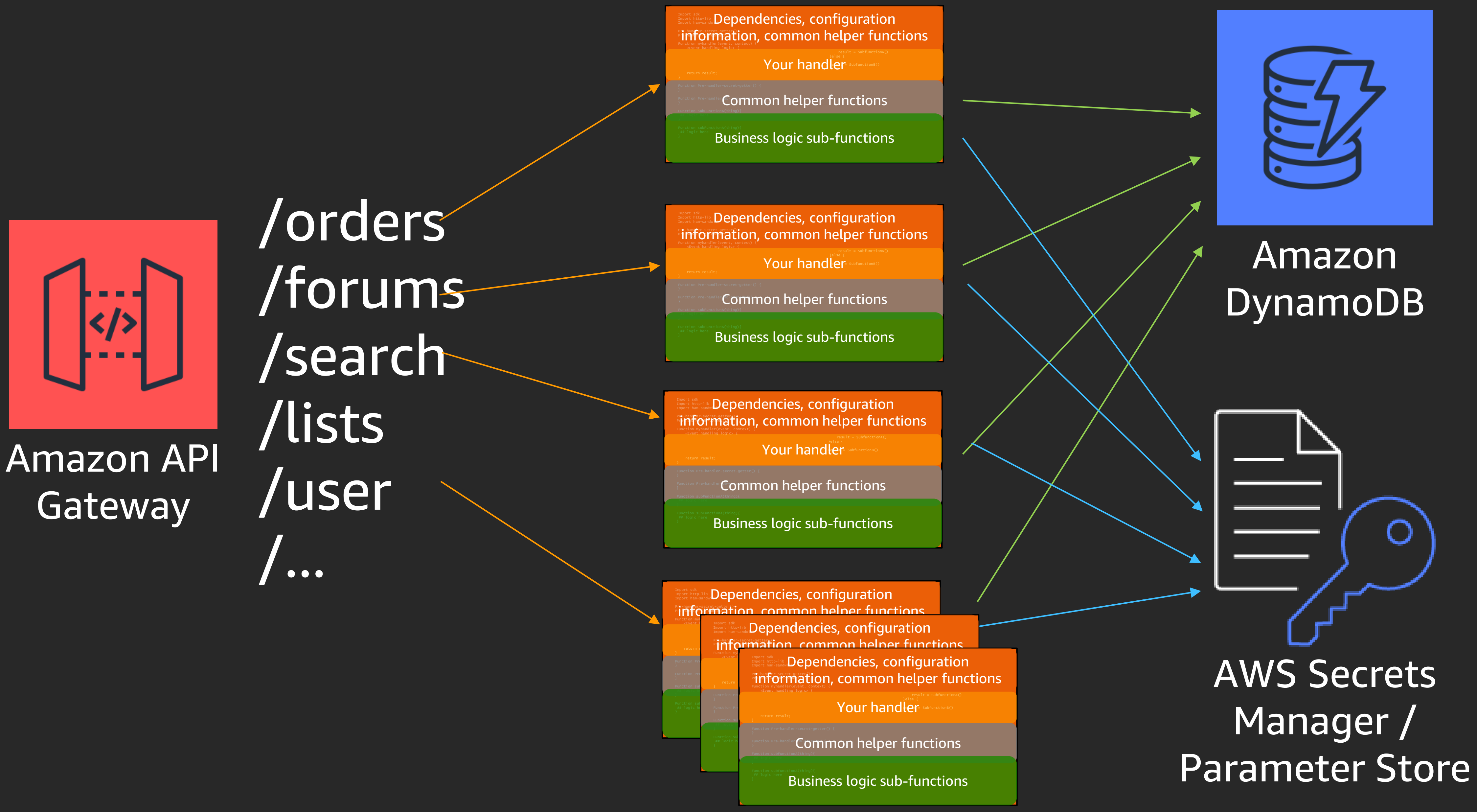
```
Function Pre-handler-secret-getter() {
}

Function Pre-handler-db-connect(){
}
```

Common helper functions

Business logic sub-functions

Anatomy of a serverless application



Amazon API Gateway

/orders
/forums
/search
/lists
/user
/…

Dependencies, configuration information, common helper functions
Your handler
Common helper functions
Business logic sub-functions

Dependencies, configuration information, common helper functions
Your handler
Common helper functions
Business logic sub-functions

Dependencies, configuration information, common helper functions
Your handler
Common helper functions
Business logic sub-functions

Dependencies, configuration information, common helper functions

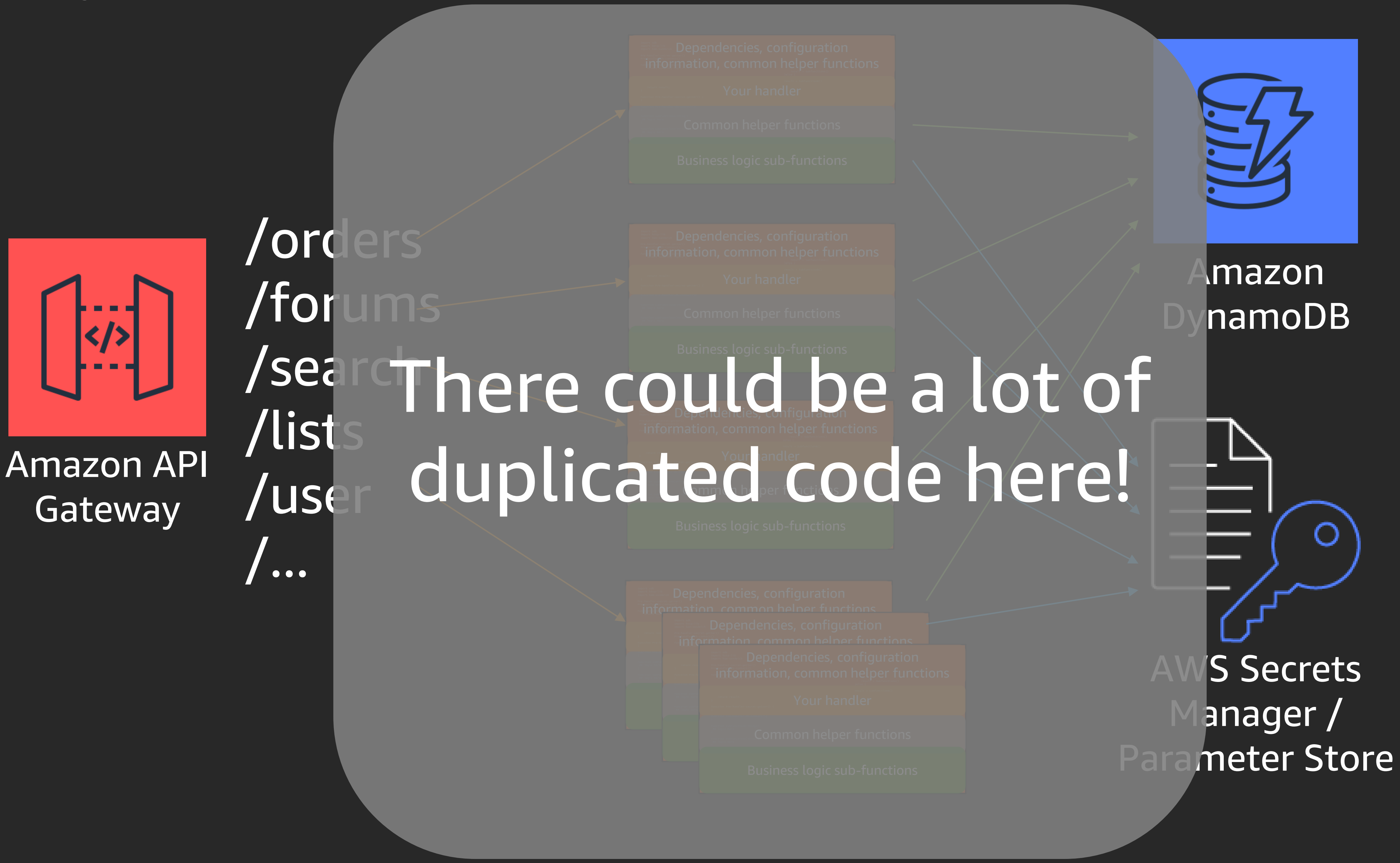
Dependencies, configuration information, common helper functions

Dependencies, configuration information, common helper functions
Your handler
Common helper functions
Business logic sub-functions

Amazon DynamoDB

AWS Secrets Manager / Parameter Store

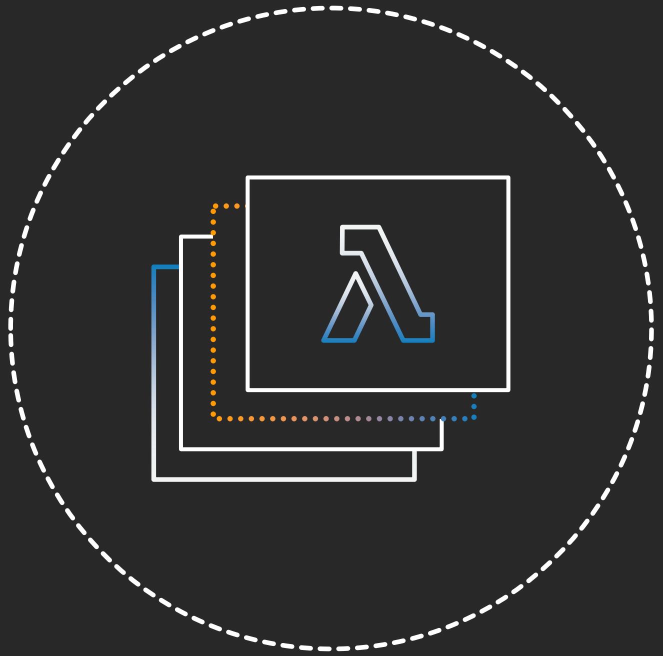
Anatomy of a serverless application



Amazon API Gateway
/orders
/forums
/search
/lists
/user
/...
Dependencies, configuration information, common helper functions
Your handler
Common helper functions
Business logic sub-functions
Amazon DynamoDB
AWS Secrets Manager / Parameter Store

There could be a lot of duplicated code here!

AWS Lambda layers



Lets functions easily share code: Upload layer once, reference within any function

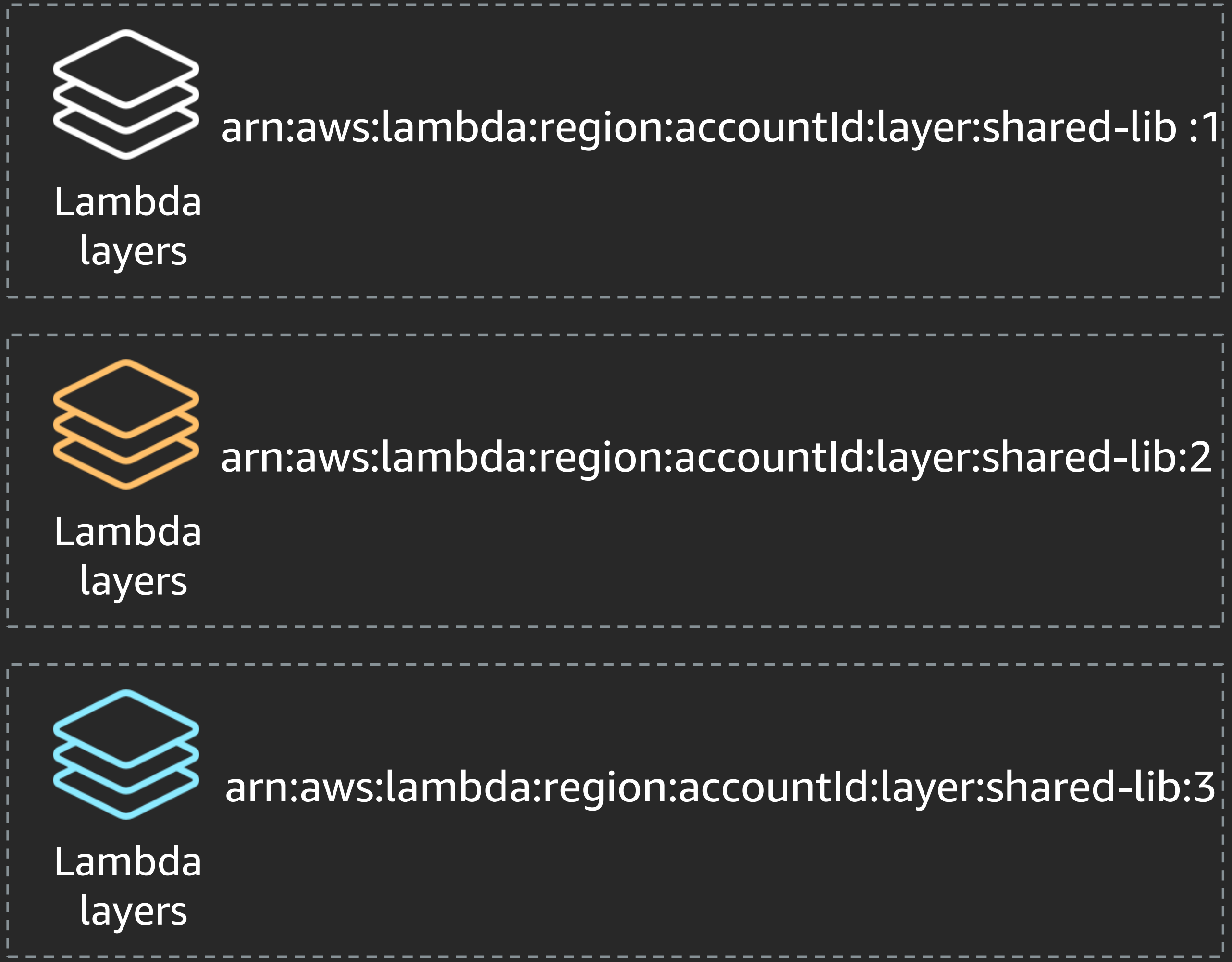
Layer can be anything: Dependencies, training data, configuration files, etc.

Promotes separation of responsibilities, lets developers iterate faster on writing business logic

Built-in support for secure sharing by ecosystem

Using AWS Lambda layers

- Put common components in a ZIP file and upload it as a Lambda layer
- Layers are immutable and can be versioned to manage updates
- When a version is deleted or permissions to use it are revoked, functions that used it previously will continue to work, but you won't be able to create new ones
- You can reference up to five layers, one of which can optionally be a custom runtime

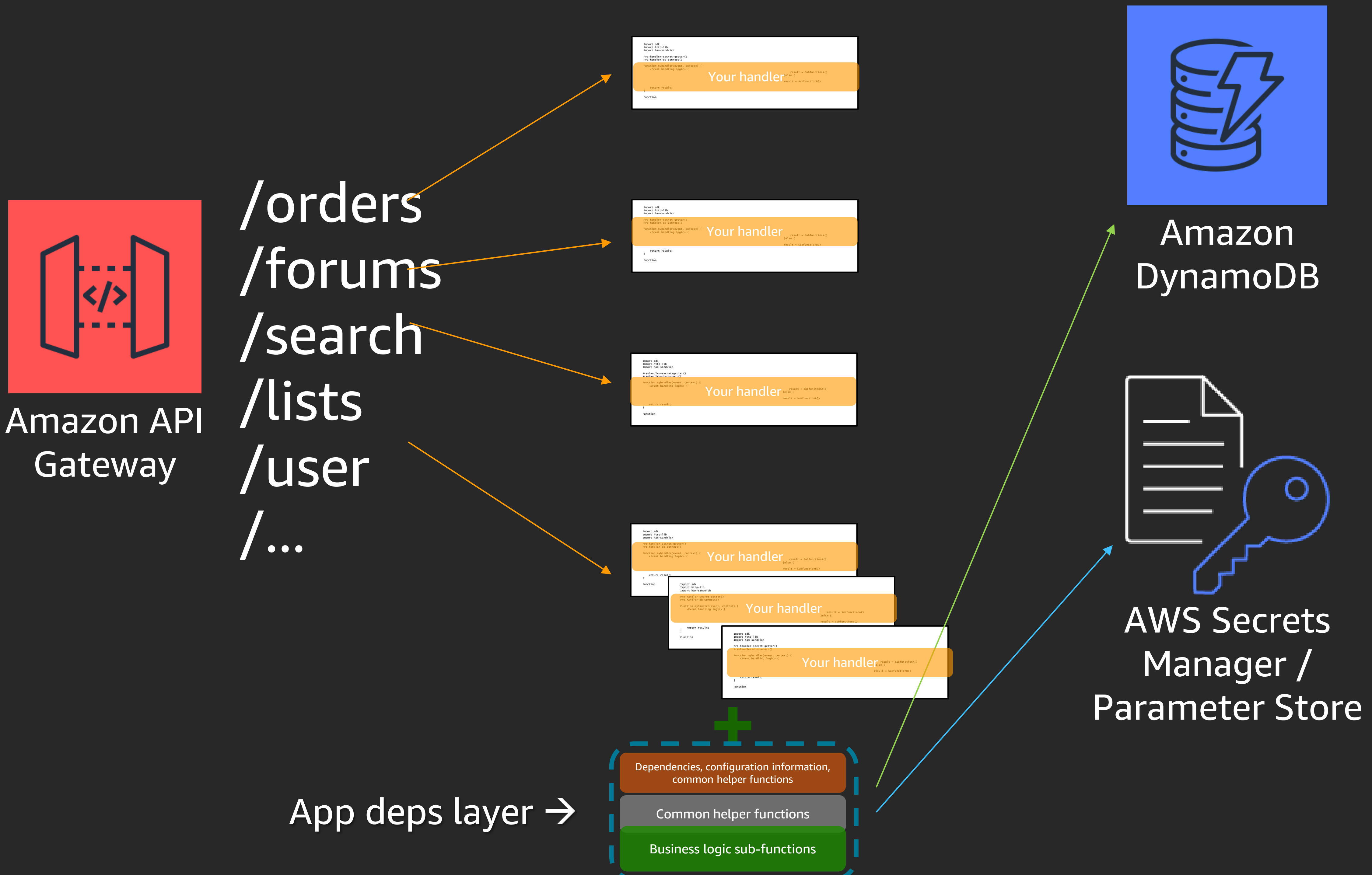


Lambda layers
arn:aws:lambda:region:accountId:layer:shared-lib :1

Lambda layers
arn:aws:lambda:region:accountId:layer:shared-lib:2

Lambda layers
arn:aws:lambda:region:accountId:layer:shared-lib:3

Anatomy of a serverless application



Amazon API Gateway

/orders
/forums
/search
/lists
/user
/…

Your handler

Your handler

Your handler

Your handler

Your handler

Your handler

App deps layer →

Dependencies, configuration information, common helper functions

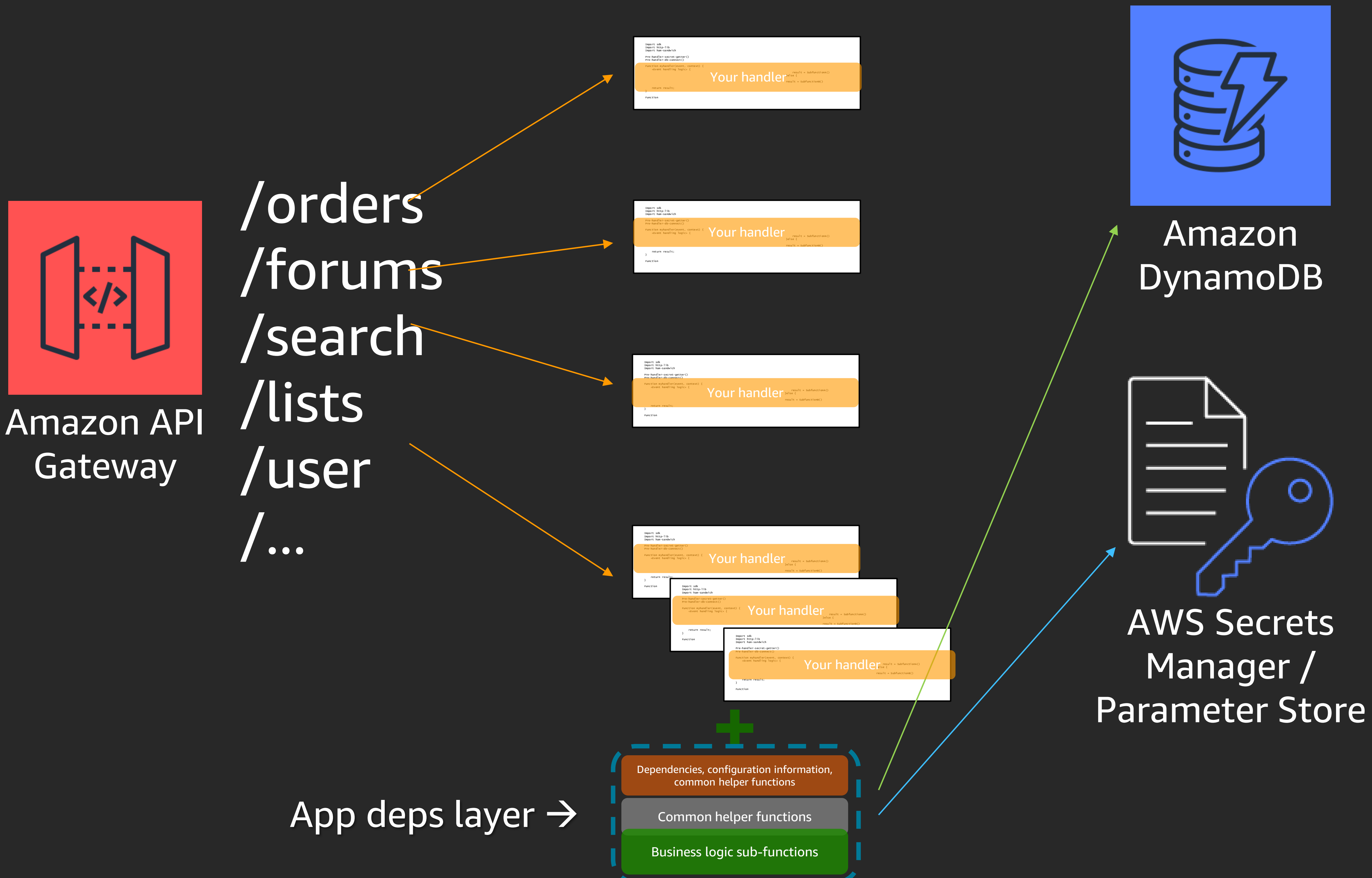
Common helper functions

Business logic sub-functions

Amazon DynamoDB

AWS Secrets Manager / Parameter Store

Anatomy of a serverless application



```
Import sdk
Import http-lib
Import ham-sandwich

Pre-handler-secret-getter()
Pre-handler-db-connect()

Function myhandler(event, context) {
    <Event handling logic> {
                                        result = SubfunctionA()
                                    }else {
                                    result = SubfunctionB()

    return result;
}

Function
```

Learn to build modern applications on AWS

Resources created by the experts at AWS to help you build and validate developer skills

Enable rapid innovation by developing your skills in designing, building, and managing modern applications

Learn to modernize your applications with free digital training and classroom offerings, including Architecting on AWS, Developing on AWS, and DevOps Engineering on AWS

Validate expertise with the AWS Certified DevOps – Professional or AWS Certified Developer – Associate exams

Visit the developer learning path at aws.amazon.com/training/path-developing

Thank you!

aws SUMMIT