aws INNOVATE

MODERN APPLICATIONS EDITION

20 October, 2022

Best practices for securing your serverless applications

Kapil Gambhir

Enterprise Solutions Architect
Amazon Internet Services Private Limited

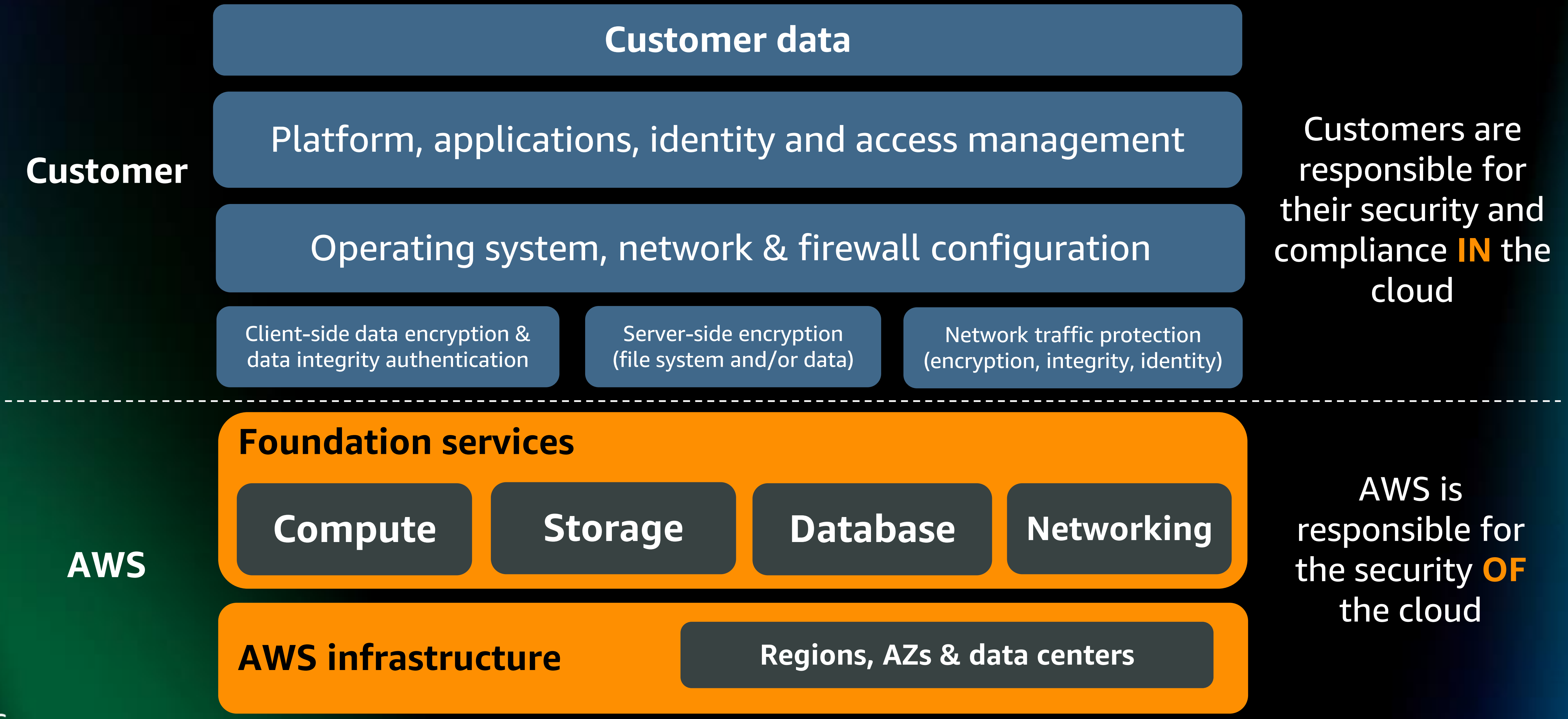
Agenda

- Overview of serverless security
- Mental model for serverless security
- Best practices for serverless applications
- Recap!

Agenda

- Overview of serverless security
- Mental model for serverless security
- Best practices for serverless applications
- Recap!

Responsibility for security and compliance is shared



Customer

Customer data

Platform, applications, identity and access management

Operating system, network & firewall configuration

Client-side data encryption & data integrity authentication

Server-side encryption (file system and/or data)

Network traffic protection (encryption, integrity, identity)

Customers are responsible for their security and compliance **IN** the cloud

AWS

Foundation services

Compute

Storage

Database

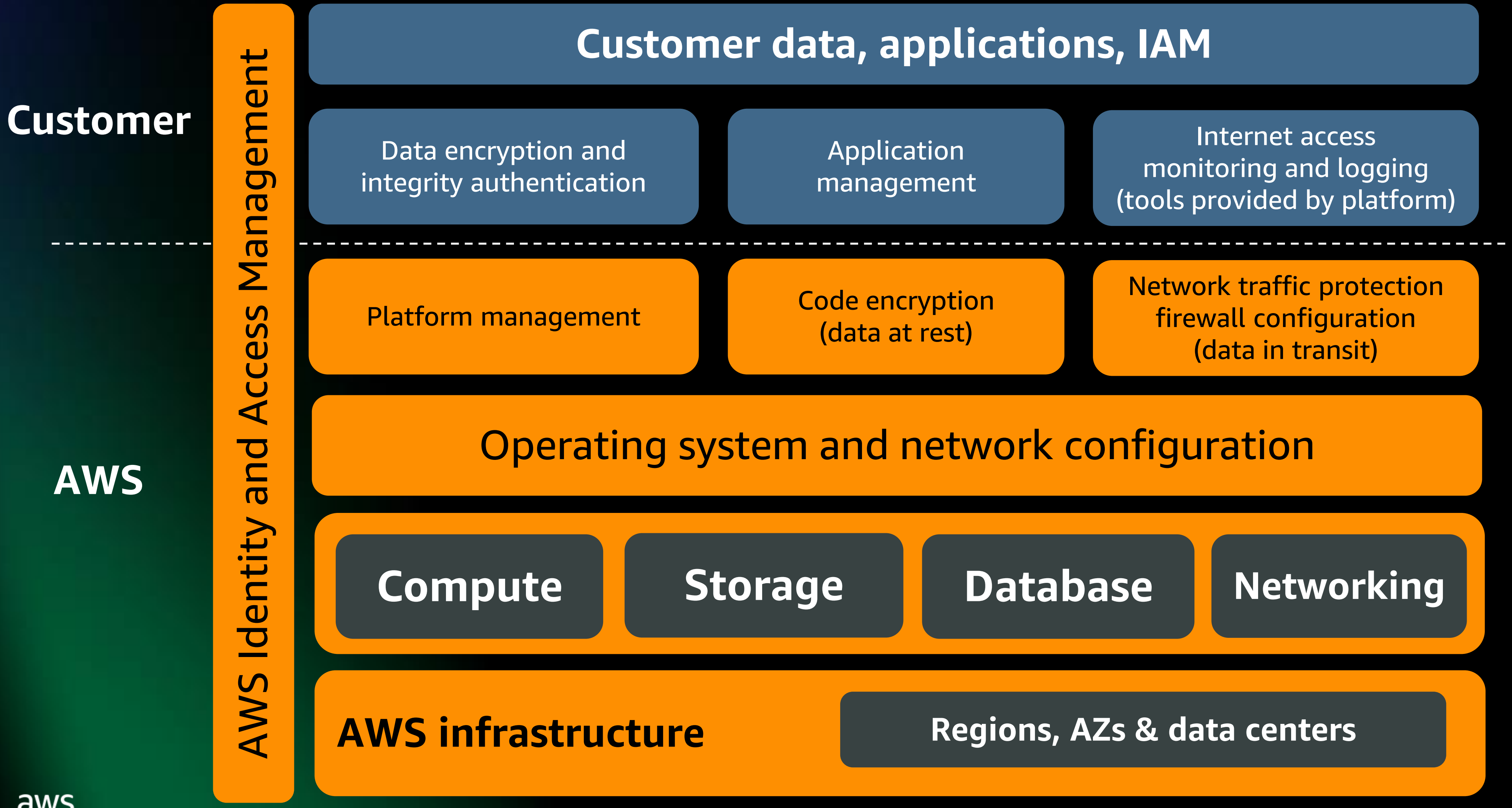
Networking

AWS infrastructure

Regions, AZs & data centers

AWS is responsible for the security **OF** the cloud

With serverless, AWS takes a greater share of the responsibility



AWS Identity and Access Management

Customer

Customer data, applications, IAM

Data encryption and integrity authentication

Application management

Internet access monitoring and logging (tools provided by platform)

AWS

Platform management

Code encryption (data at rest)

Network traffic protection firewall configuration (data in transit)

Operating system and network configuration

Compute

Storage

Database

Networking

AWS infrastructure

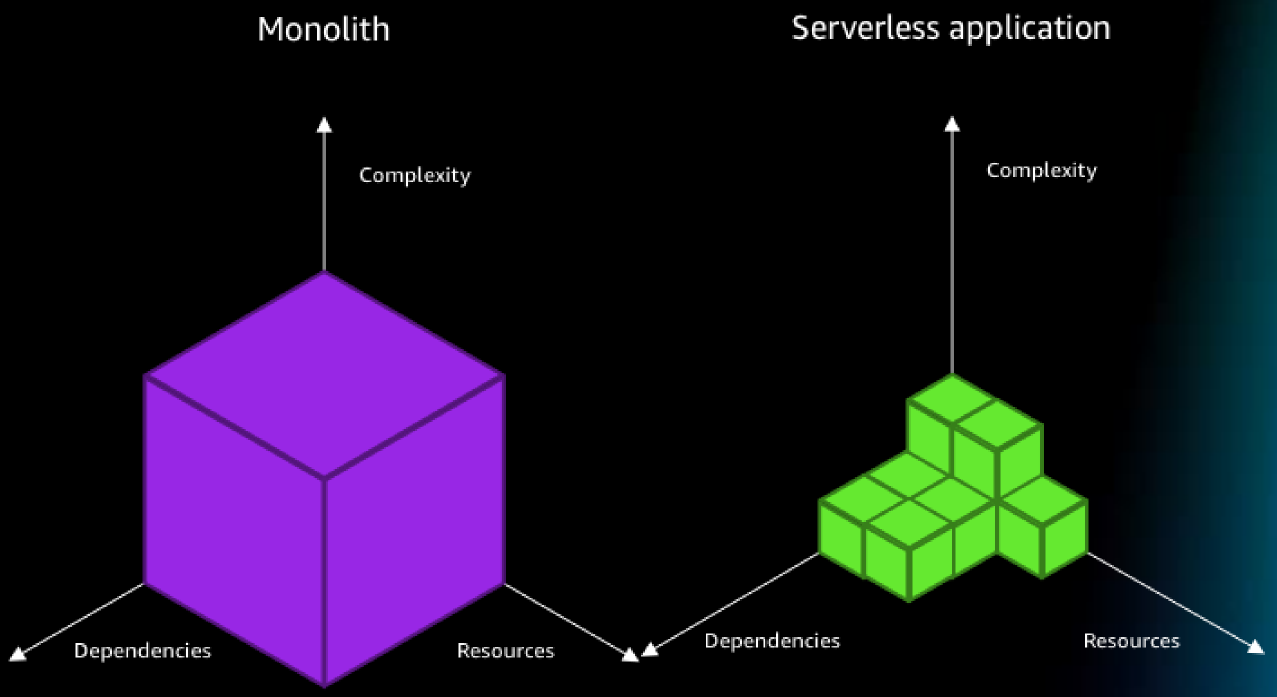
Regions, AZs & data centers

Agenda

- Overview of serverless security
- Mental model for serverless security
- Best practices for serverless applications
- Recap!

Smaller units allow more fine-grained control

- Microservices are less complex and have smaller scope
- Take advantage of these benefits by:
  - Limiting scope of permissions
  - Avoiding monolithic functions
  - Considering data access needs
  - Excluding unnecessary dependencies



OWASP Serverless Top 10

- S1:2017 Injection
- S2:2017 Broken Authentication
- S3:2017 Sensitive Data Exposure
- S4:2017 XML External Entities (XXE)
- S5:2017 Broken Access Control
- S6:2017 Security Misconfiguration
- S7:2017 Cross-Site Scripting (XSS)
- S8:2017 Insecure Deserialization
- S9:2017 Using Components with Known Vulnerabilities
- S10:2017 Insufficient Logging and Monitoring

Open Web Application Security Project® (*OWASP*)

s12d.com/owasp-top10

AWS Well-Architected Framework design principles

- Implement a strong identity foundation
- Enable traceability
- Apply security at all layers
- Automate security best practices
- Protect data in transit and at rest
- Keep people away from data
- Prepare for security events

s12d.com/well-arch-security
s12d.com/serverless-lens

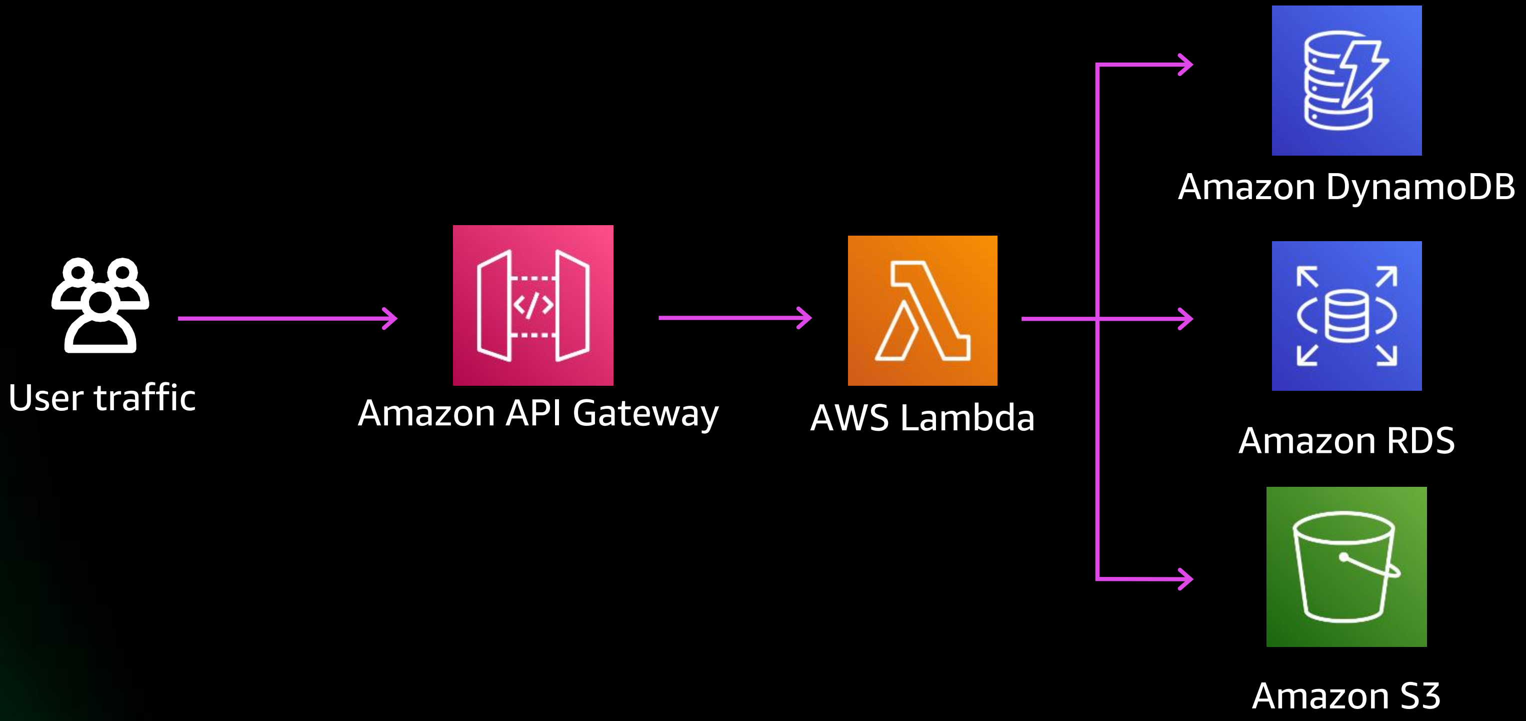
Serverless security best practices

- Use authentication and authorization mechanisms
- Data encryption and integrity
- Monitoring, logging, and configuration management
- Denial of service and infrastructure protection

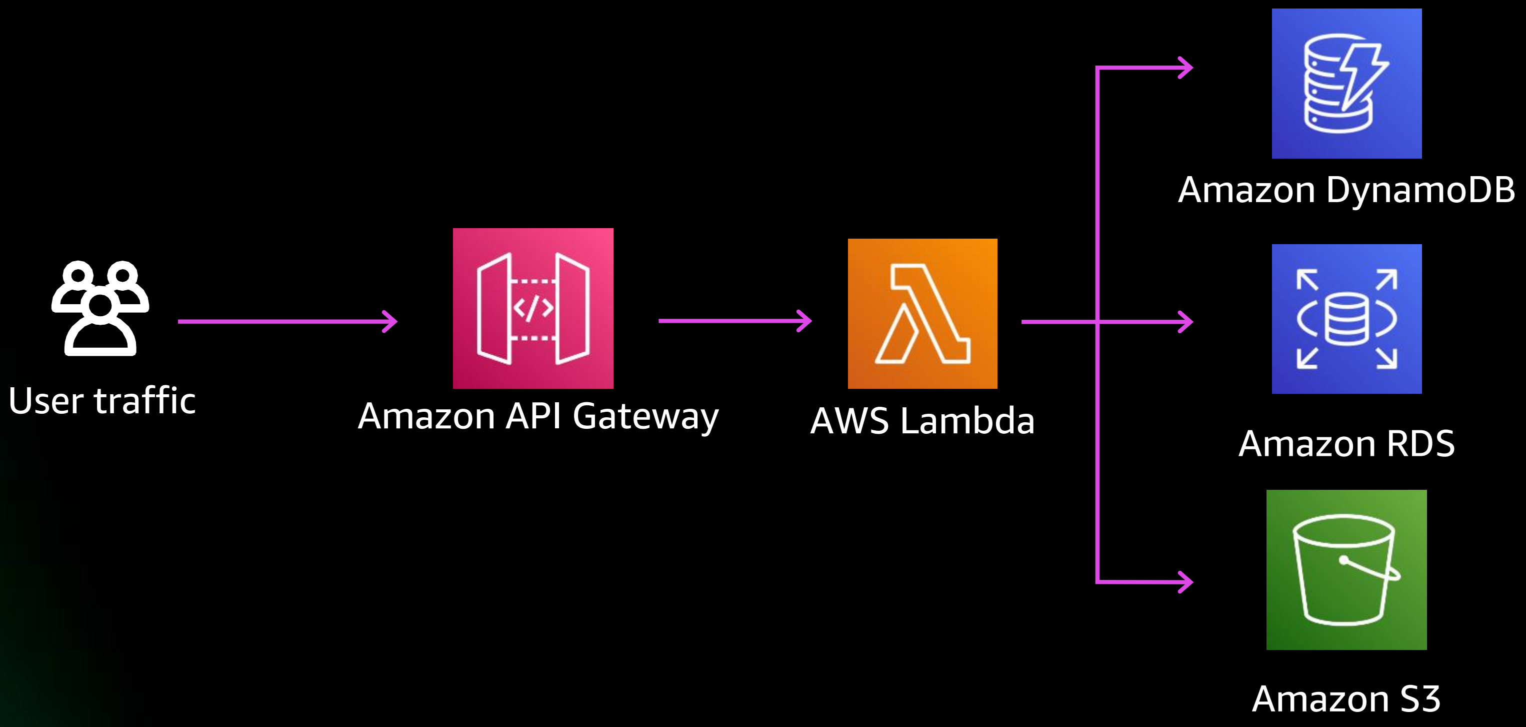
Agenda

- Overview of serverless security
- Mental model for serverless security
- Best practices for serverless applications
- Recap!

Candidate Architecture



Best practice 1: Use authentication and authorization mechanisms



Best practice 1: What to consider

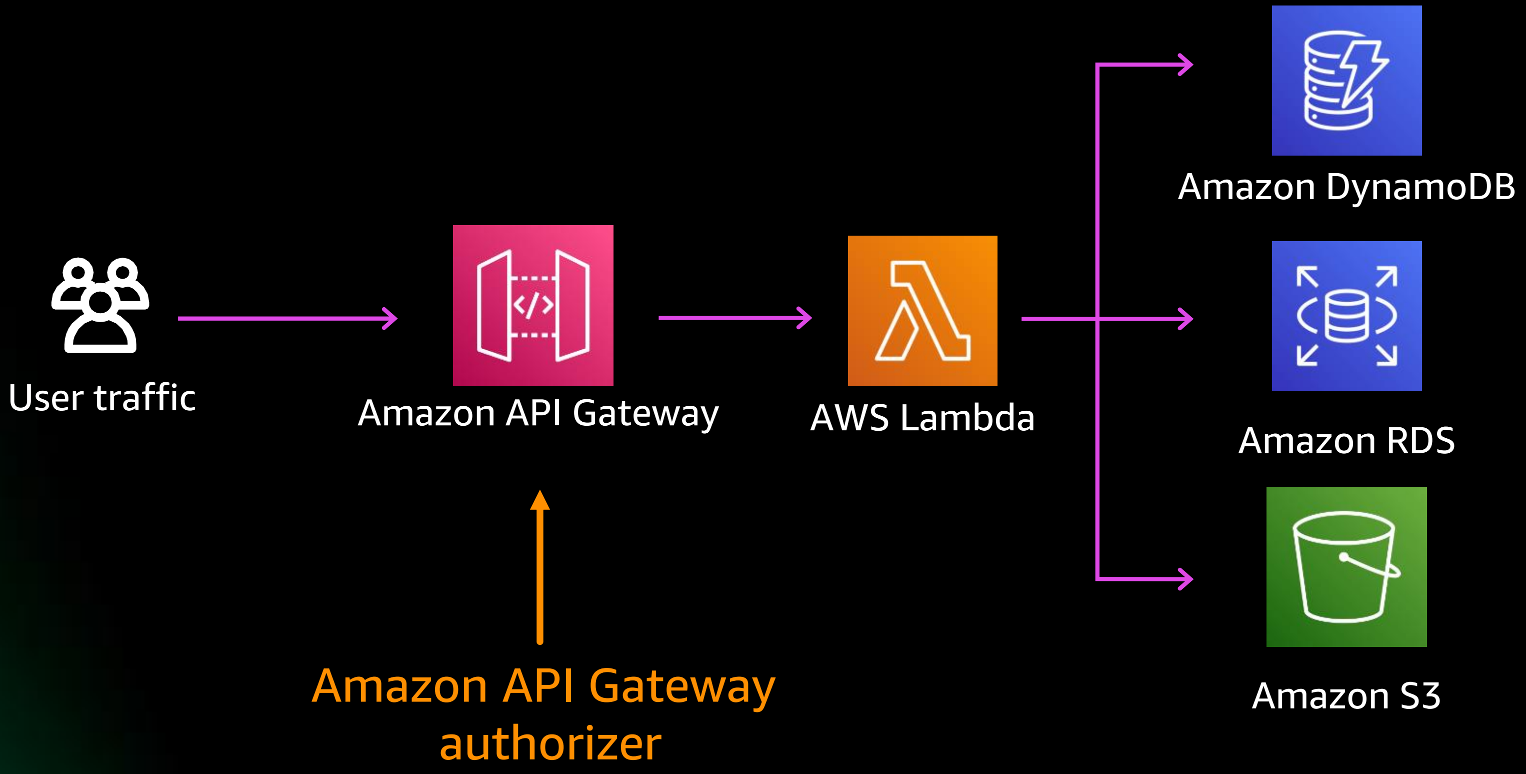
USE AUTHENTICATION AND AUTHORIZATION MECHANISMS

- Use appropriate authentication and authorization mechanisms
- Follow least-privilege model
- Take advantage of smaller, single purpose lambda functions
- Store secrets securely

OWASP Serverless Top 10
S2:2017 Broken Authentication
S5:2017 Broken Access Control

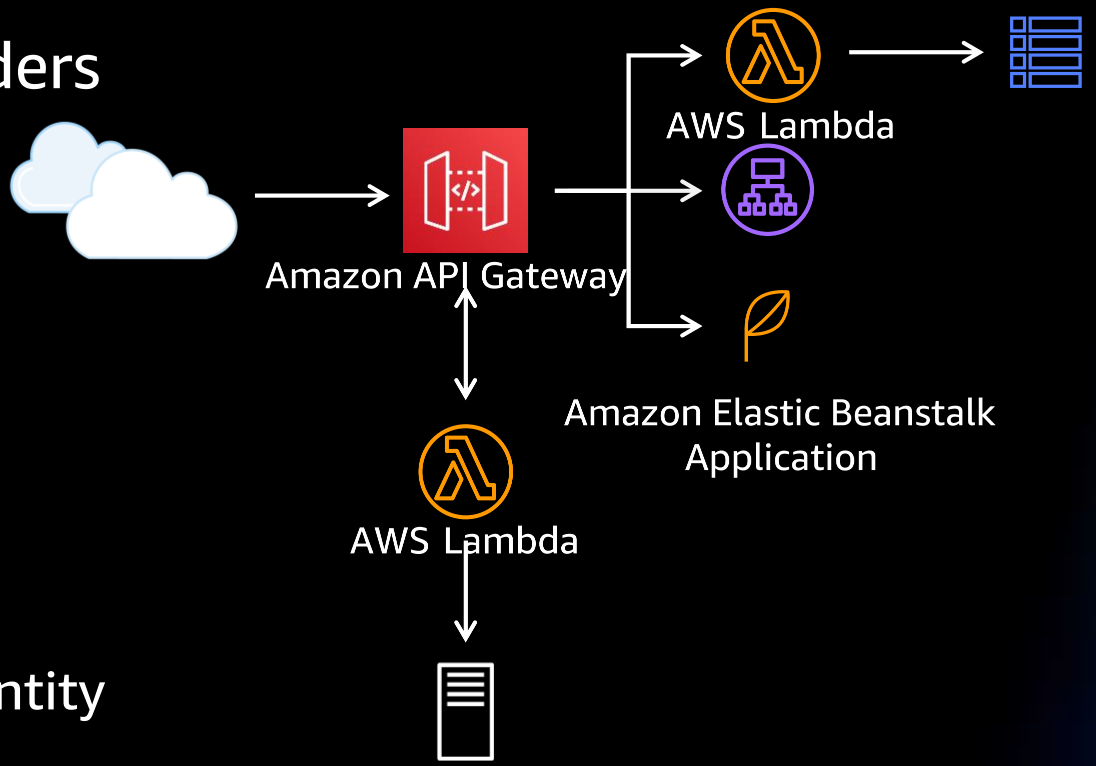
AWS Well-Architected Framework
Implement a strong identity foundation
Apply security at all layers

Best practice 1: Authentication and authorization

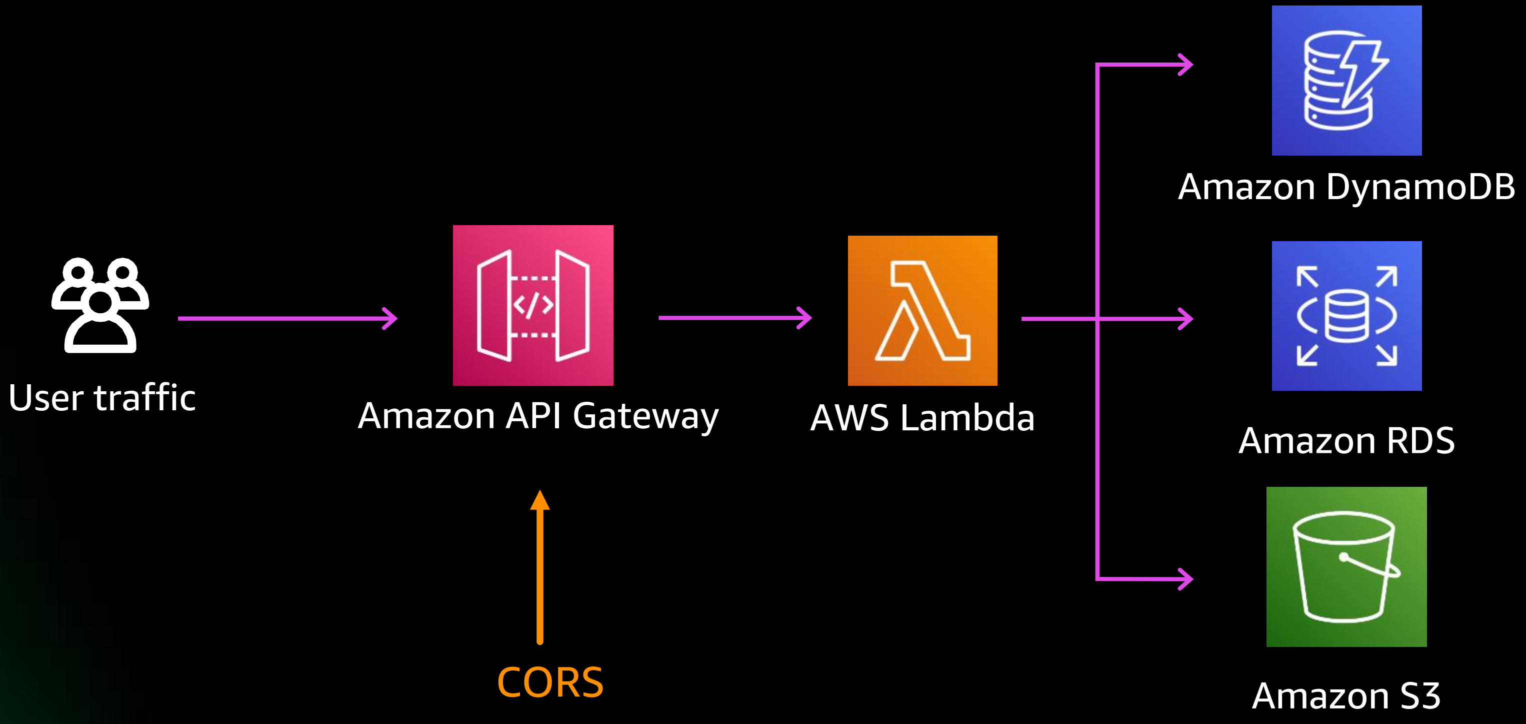


Best practice 1: Authentication and authorization

- Amazon API Gateway supports **multiple mechanisms** to control access (fine-grained)
- Integrate with variety of identity providers
  - Amazon Cognito user pools
  - OIDC, OAuth (JWT authorizer)
  - IAM
  - Lambda authorizer (custom)
- Use mutual TLS (**mTLS**) for B2B service2service authentication
  - Client presents X.509 certificate to prove identity

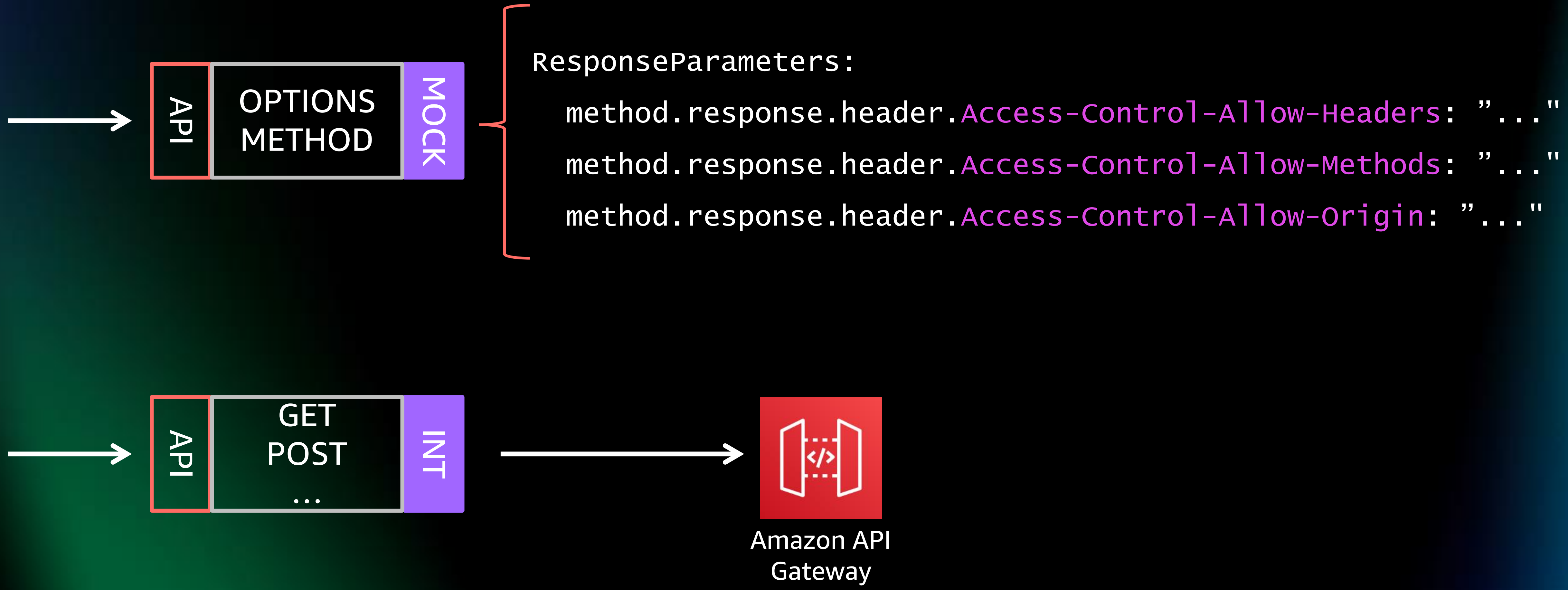


Best practice 1: More access control



Best practice 1: More access control

CROSS-ORIGIN RESOURCE SHARING (CORS)

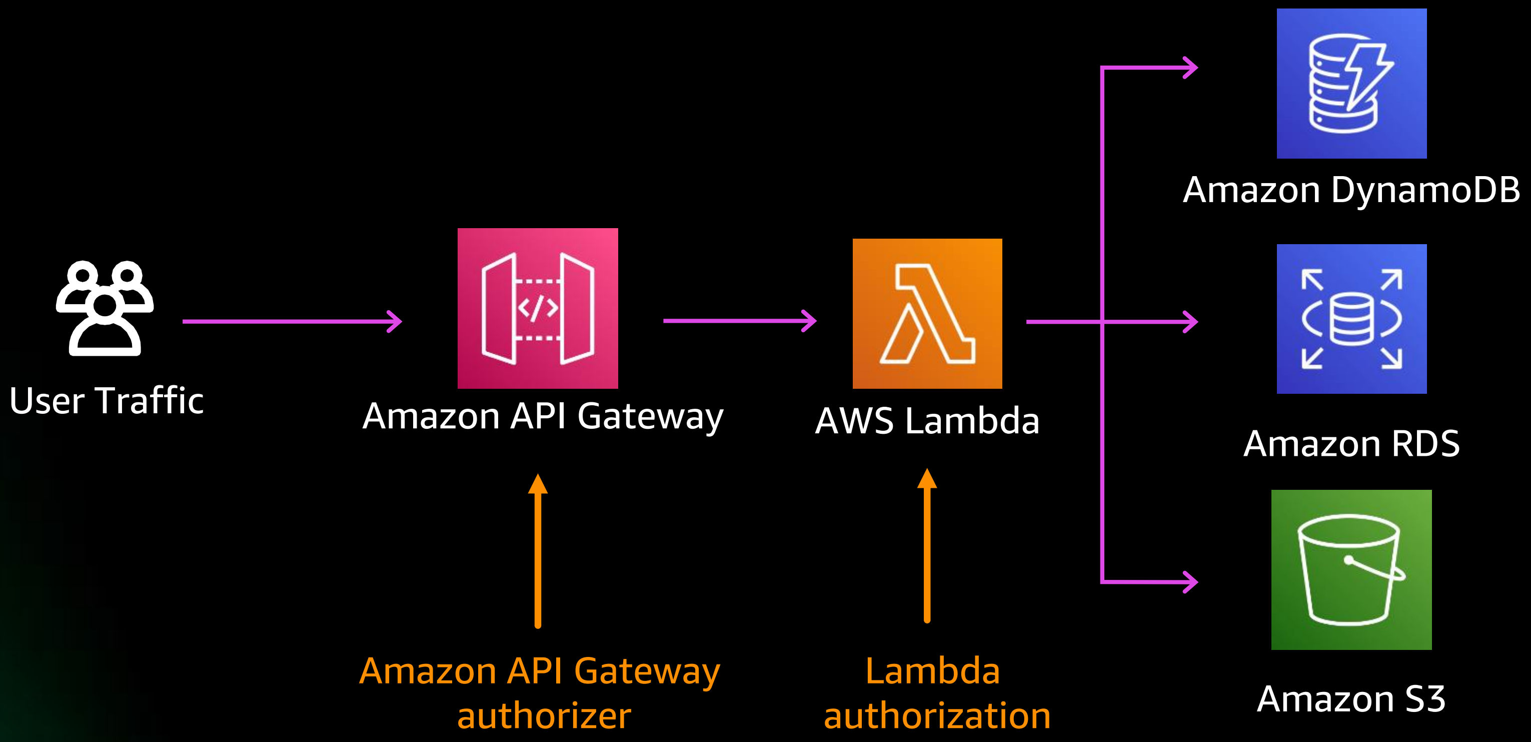


```
ResponseParameters:
  method.response.header.Access-Control-Allow-Headers: ”..."
  method.response.header.Access-Control-Allow-Methods: ”..."
  method.response.header.Access-Control-Allow-Origin: ”..."
```

https://cors.serverlessland.com/

Best practice 1: Authentication and authorization

LAMBDA AUTHORIZATION



Secure Lambda functions with IAM

Function policy

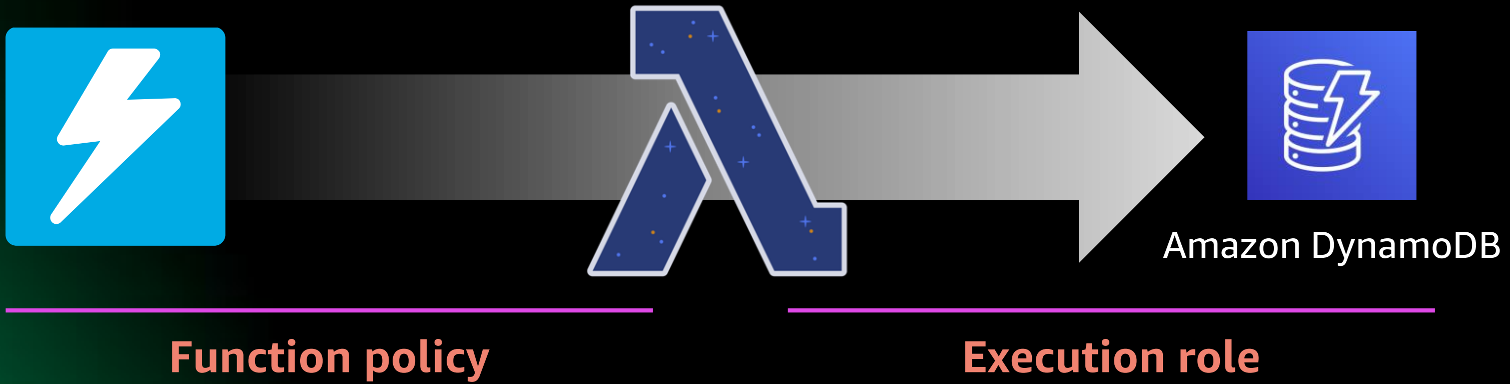
- Defines how function can be invoked
- Supports cross-account access
- Used for synchronous and asynchronous invocations

"Actions on API Gateway A can invoke Lambda function B"

Execution role

- Defines which AWS resources can access via IAM
- Used for poll-based invocations (Lambda polling)

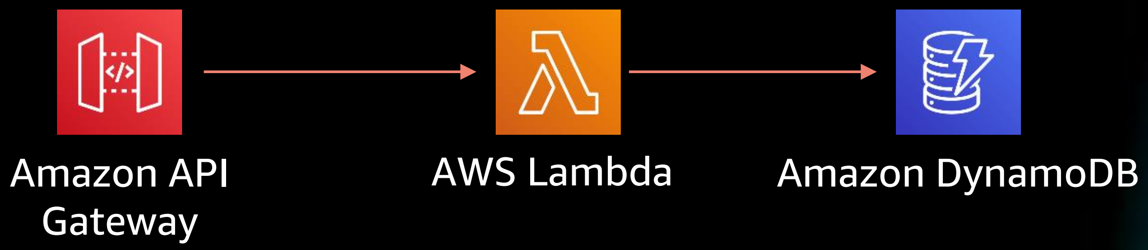
"Lambda function A can write data to DynamoDB Table B."

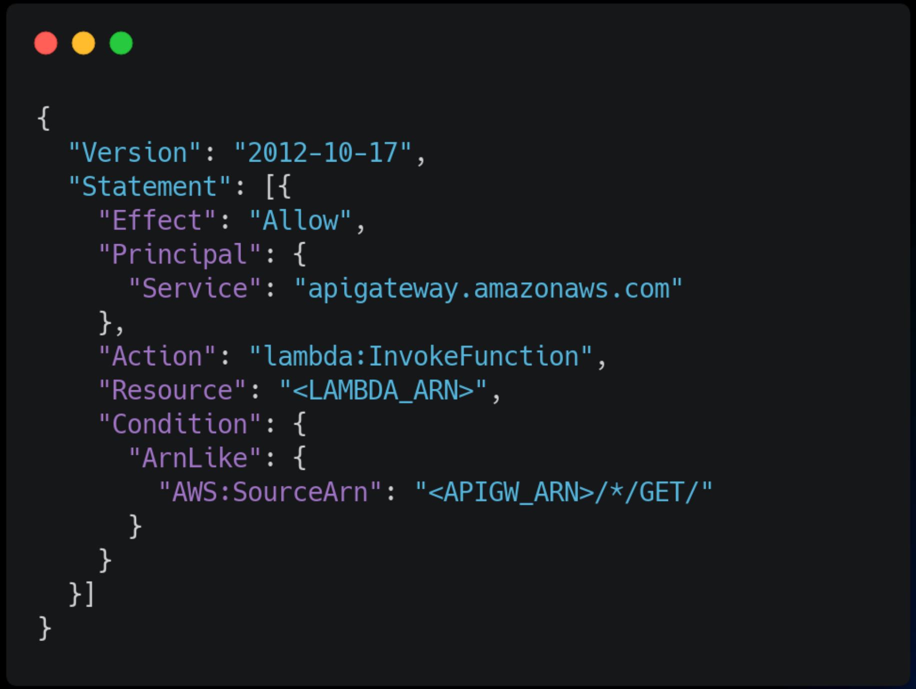


Best practice 1: Authentication and authorization

FUNCTION POLICY: INVOKING LAMBDA FUNCTION

- Invoking a function requires Resource based or Identity based permission.
- Allows synchronous and asynchronous event sources to invoke function
  - API Gateway to method level
- AWS Console, AWS SAM, and others tools may update automatically



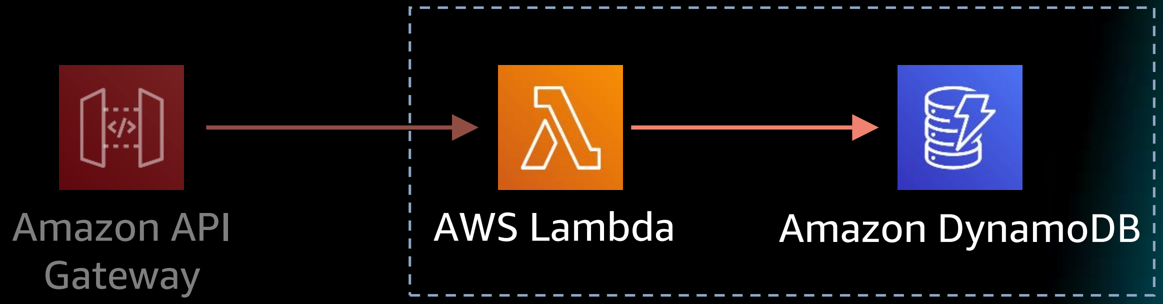


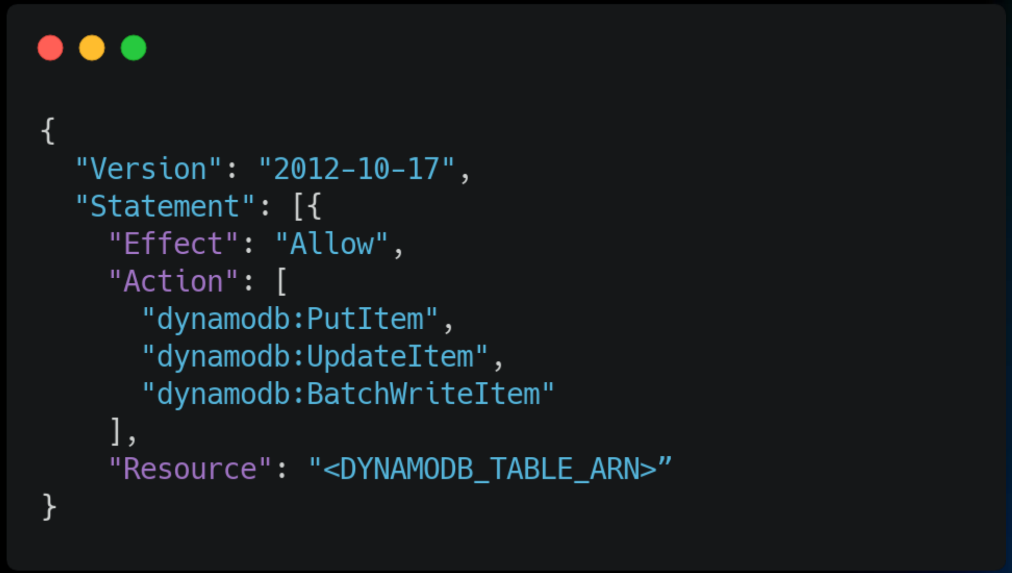
```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Principal": {
      "Service": "apigateway.amazonaws.com"
    },
    "Action": "lambda:InvokeFunction",
    "Resource": "<LAMBDA_ARN>",
    "Condition": {
      "ArnLike": {
        "AWS:SourceArn": "<APIGW_ARN>/*/GET/"
      }
    }
  }]
}
```

Best practice 1: Authentication and authorization

LAMBDA EXECUTION ROLE: ALLOWS ACCESS TO AWS RESOURCES

- **Created explicitly** as part of function development – AWS SAM and other tools may assist with a basic policy
- Best practices
  - Define a **unique policy per function**
  - Apply **principles of least privilege**
  - Avoid wildcard permissions
  - Don’t forget Amazon CloudWatch, AWS X-Ray (if desired)



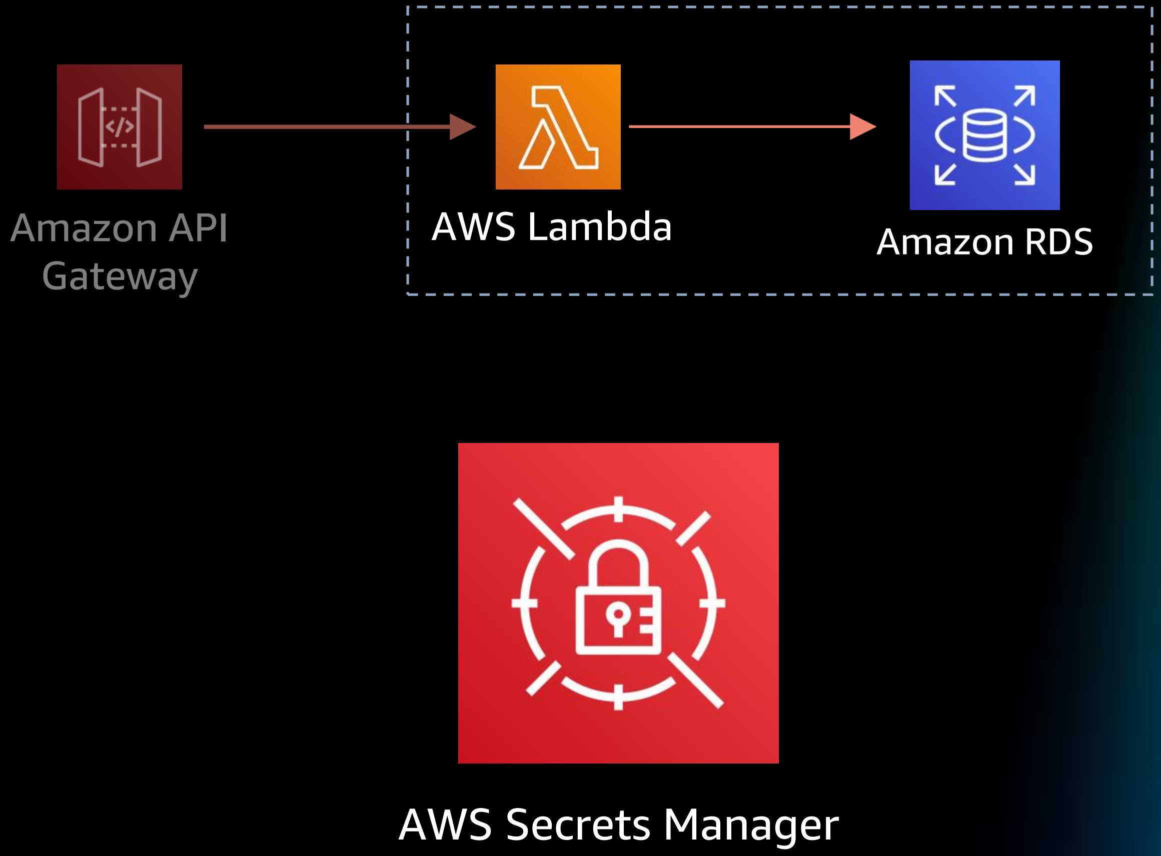


```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Action": [
      "dynamodb:PutItem",
      "dynamodb:UpdateItem",
      "dynamodb:BatchWriteItem"
    ],
    "Resource": "<DYNAMODB_TABLE_ARN>”
}
```

Best practice 1: Authentication and authorization

SECRETS MANAGEMENT

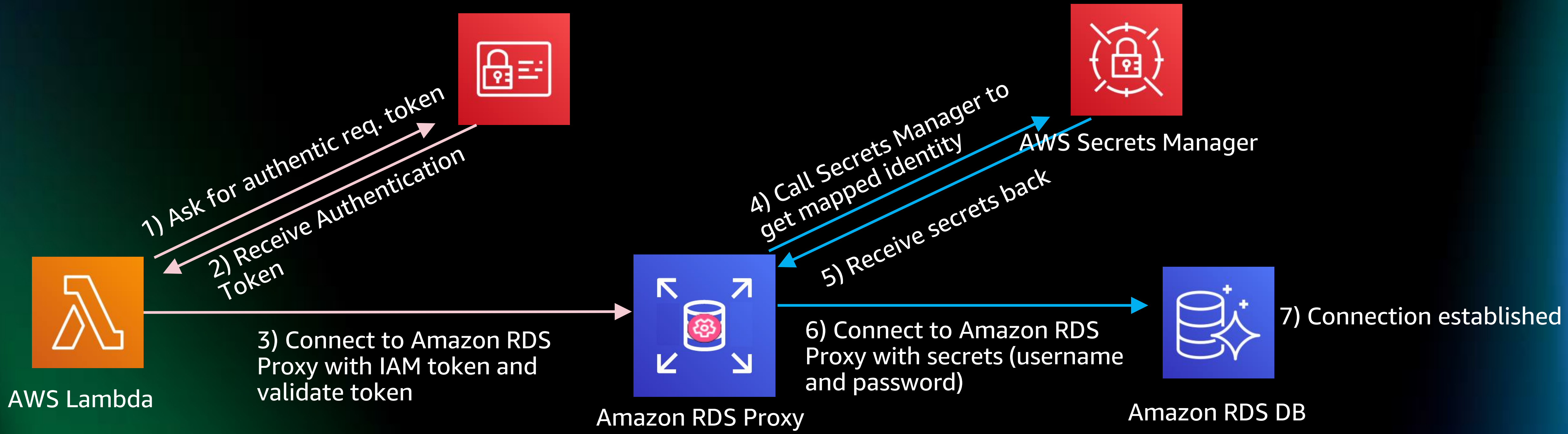
- Function requires static, sensitive data (e.g., API key or password)
- While convenient, do **not** use Lambda environment variables – accessible to anyone with access to the function
- Answer: Use **purpose-built services**, such as **AWS Secrets Manager** secured with IAM permissions



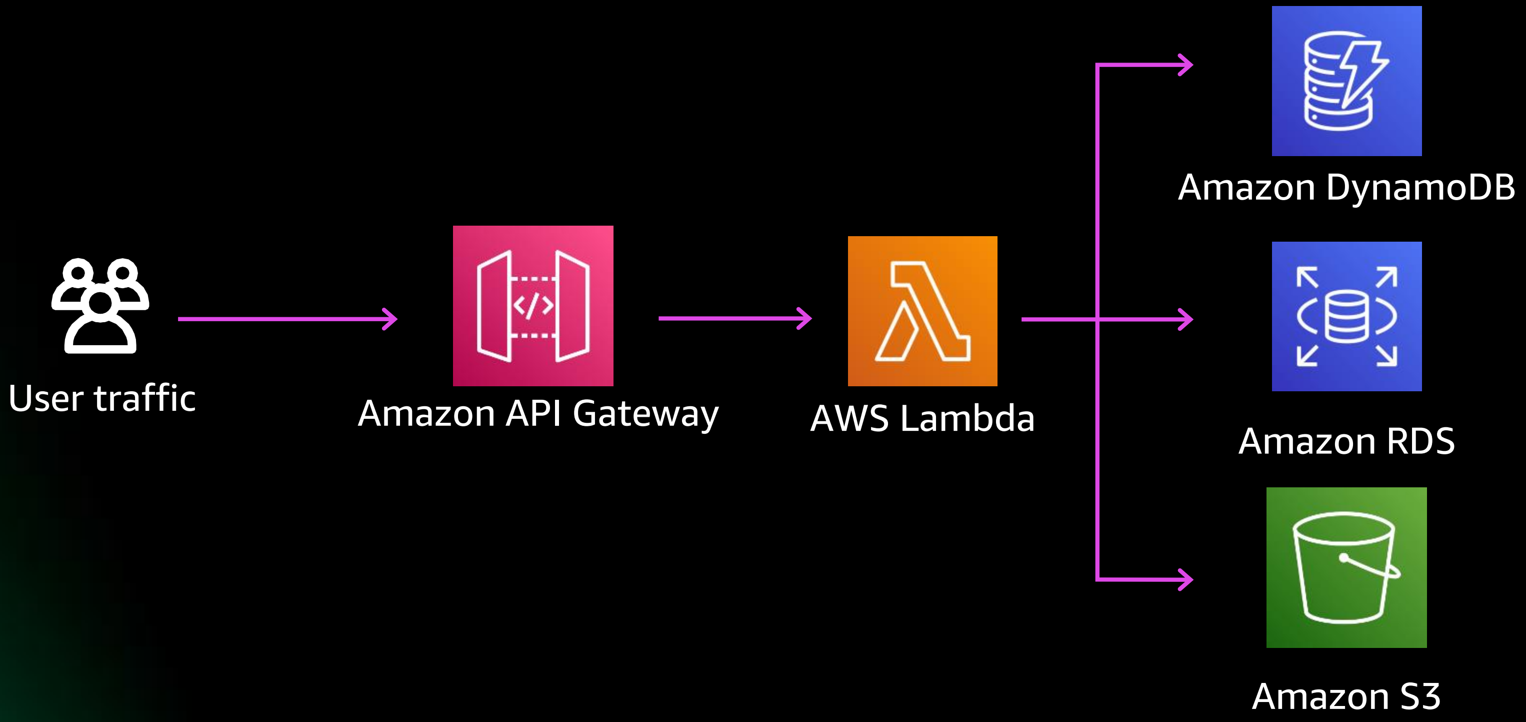
Best practice 1: Authentication and authorization

USE AMAZON RDS PROXY TO ENFORCE IAM AUTHENTICATION WITH YOUR RELATIONAL DATABASES

Centrally manage database credentials using Secrets Manager


1) Ask for authentic req. token
2) Receive Authentication Token
4) Call Secrets Manager to get mapped identity
AWS Secrets Manager
5) Receive secrets back
AWS Lambda
3) Connect to Amazon RDS Proxy with IAM token and validate token
Amazon RDS Proxy
6) Connect to Amazon RDS Proxy with secrets (username and password)
7) Connection established
Amazon RDS DB

Best practice 2: Data encryption and integrity



Best practice 2: What to consider

DATA ENCRYPTION AND INTEGRITY

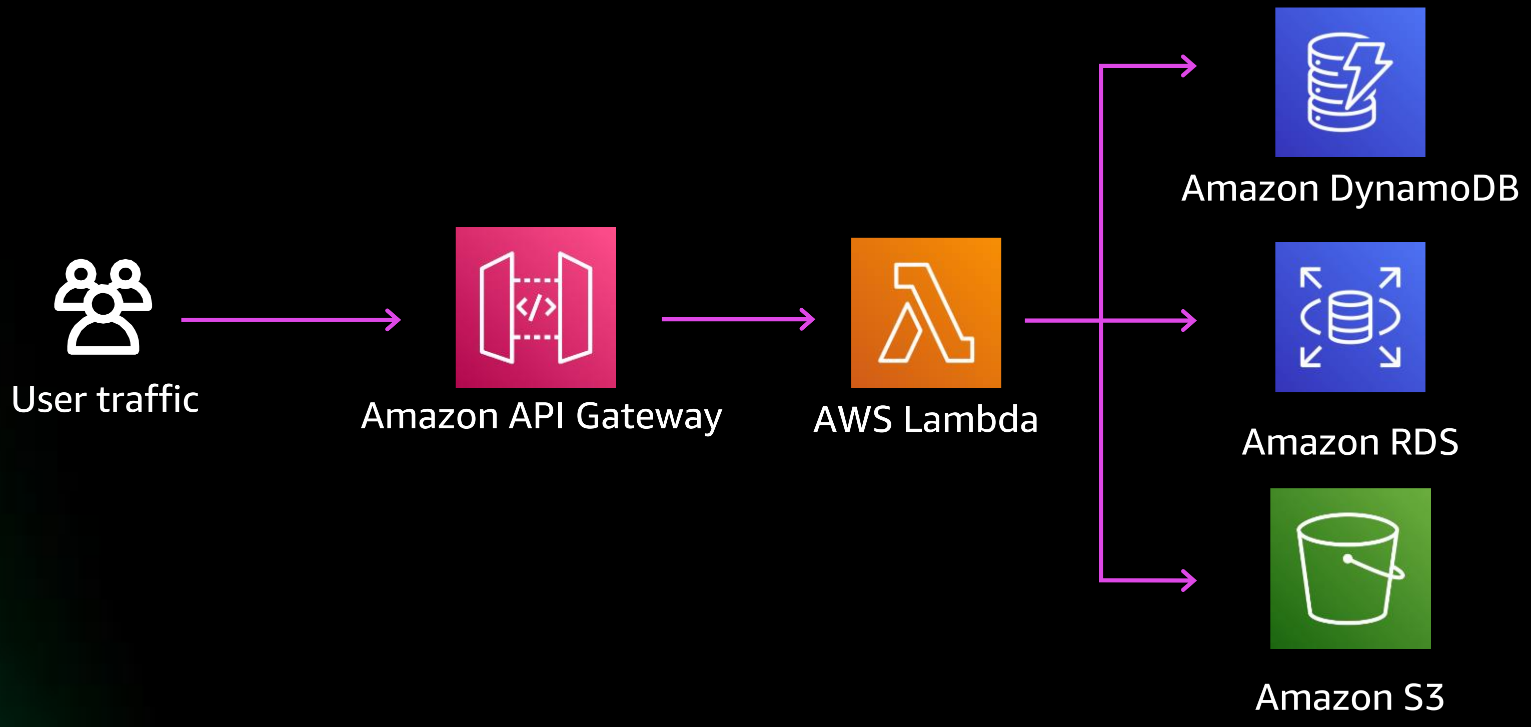
- Identify and classify sensitive data
- Minimize storage of sensitive data to only what is necessary
- Protect data at rest and in transit
- Use infrastructure provider services for key management and encryption of stored data, secrets, and environment variables
- Protect against common web exploits (e.g., XSS, SQL injection)
- Follow secure coding practices

OWASP Serverless Top 10
S1:2017 Injection
S3:2017 Sensitive Data Exposure
S8:2017 Insecure Deserialization

AWS Well-Architected Framework
Protect data in transit and at rest
Apply security at all layers

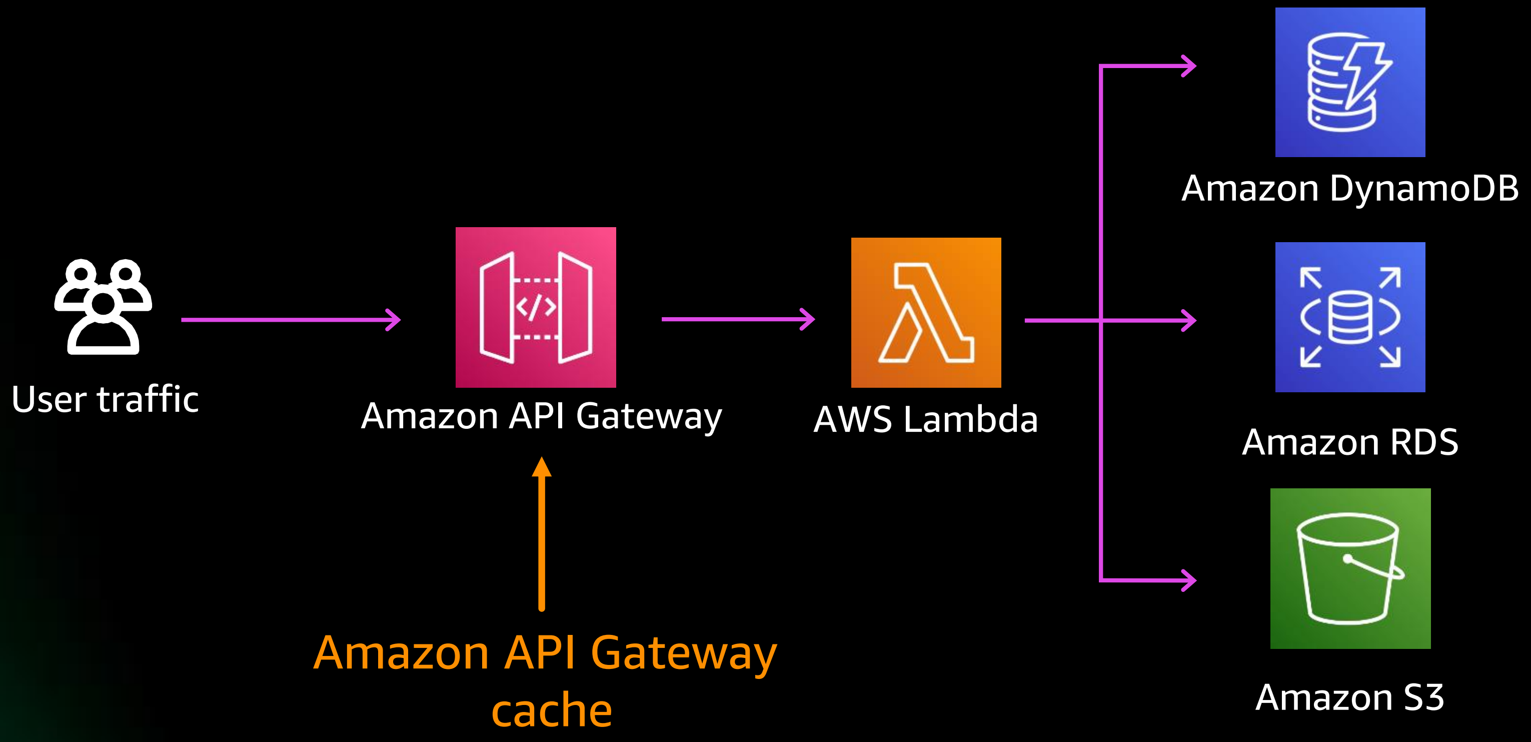
Best practice 2: Data encryption and integrity

PROTECT DATA AT REST



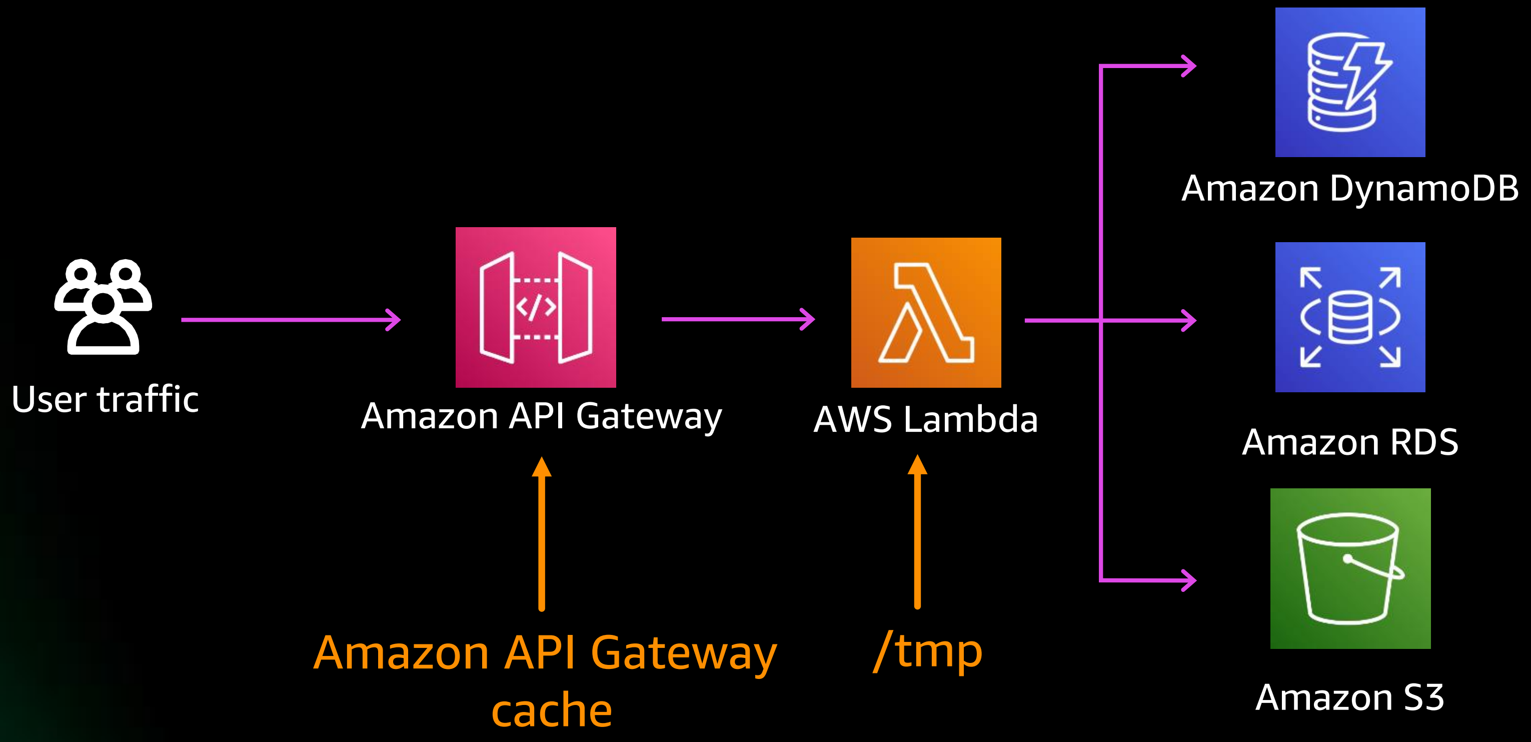
Best practice 2: Data encryption and integrity

PROTECT DATA AT REST



Best practice 2: Data encryption and integrity

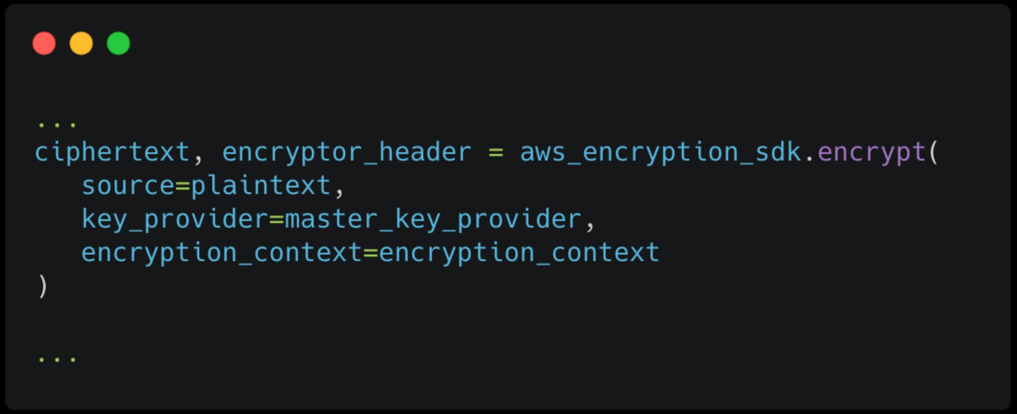
PROTECT DATA AT REST



Best practice 2: Data encryption and integrity

AWS ENCRYPTION SDK

- APIs and **data format** for the encryption
- Interface simplified with AWS KMS for the **encryption in envelope**
- **Open-source**, open-specification, Apache 2.0
- Multiple languages
  - **AWS Encryption SDK for Java**
  - **AWS Encryption SDK for Python**
  - **AWS Encryption SDK for C**
  - **AWS Encryption SDK for JavaScript and Node.js**
- **Multiple** KMS keys and **data key caching** built in



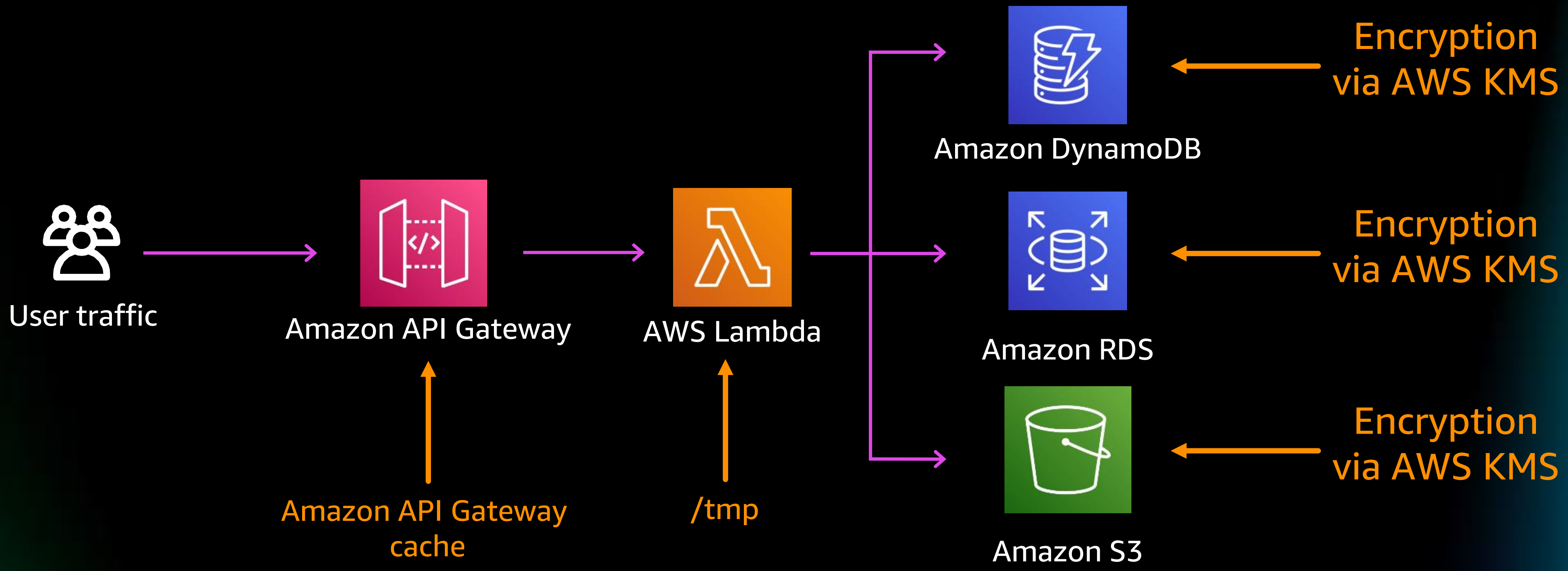
```
...
ciphertext, encryptor_header = aws_encryption_sdk.encrypt(
    source=plaintext,
    key_provider=master_key_provider,
    encryption_context=encryption_context
)

...
```

s12d.com/encryption-sdk-python

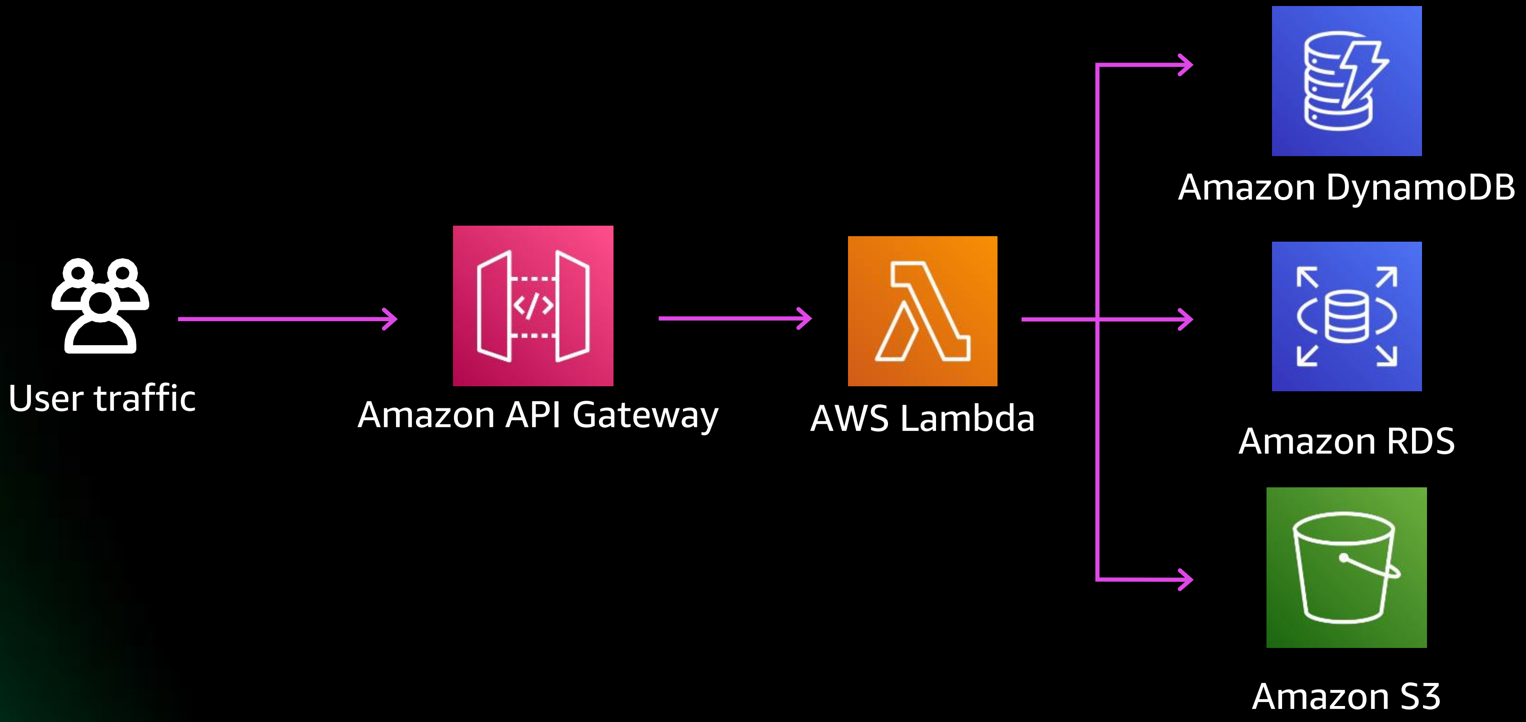
Best practice 2: Data encryption and integrity

PROTECT DATA AT REST



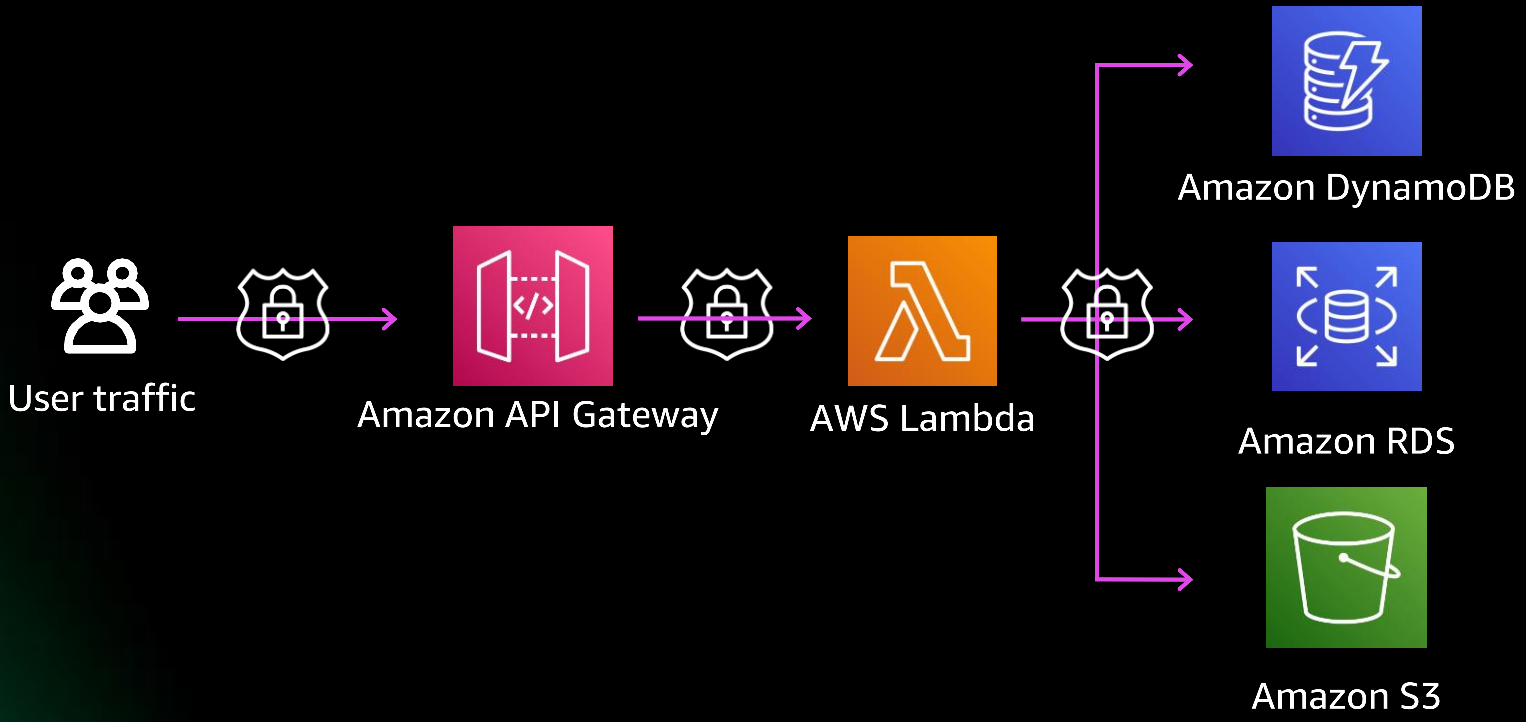
Best practice 2: Data encryption and integrity

PROTECT DATA IN TRANSIT



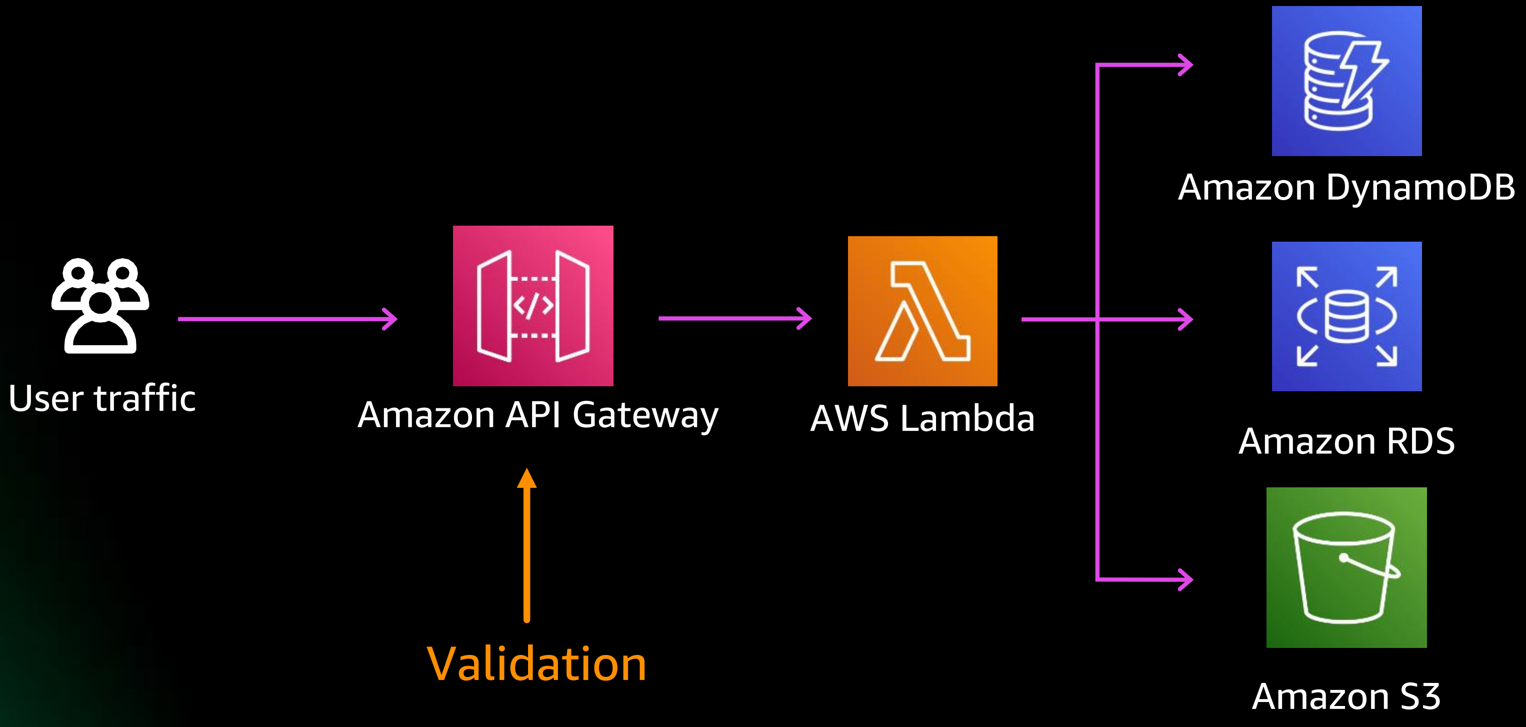
Best practice 2: Data encryption and integrity

PROTECT DATA IN TRANSIT



Best practice 2: Data encryption and integrity

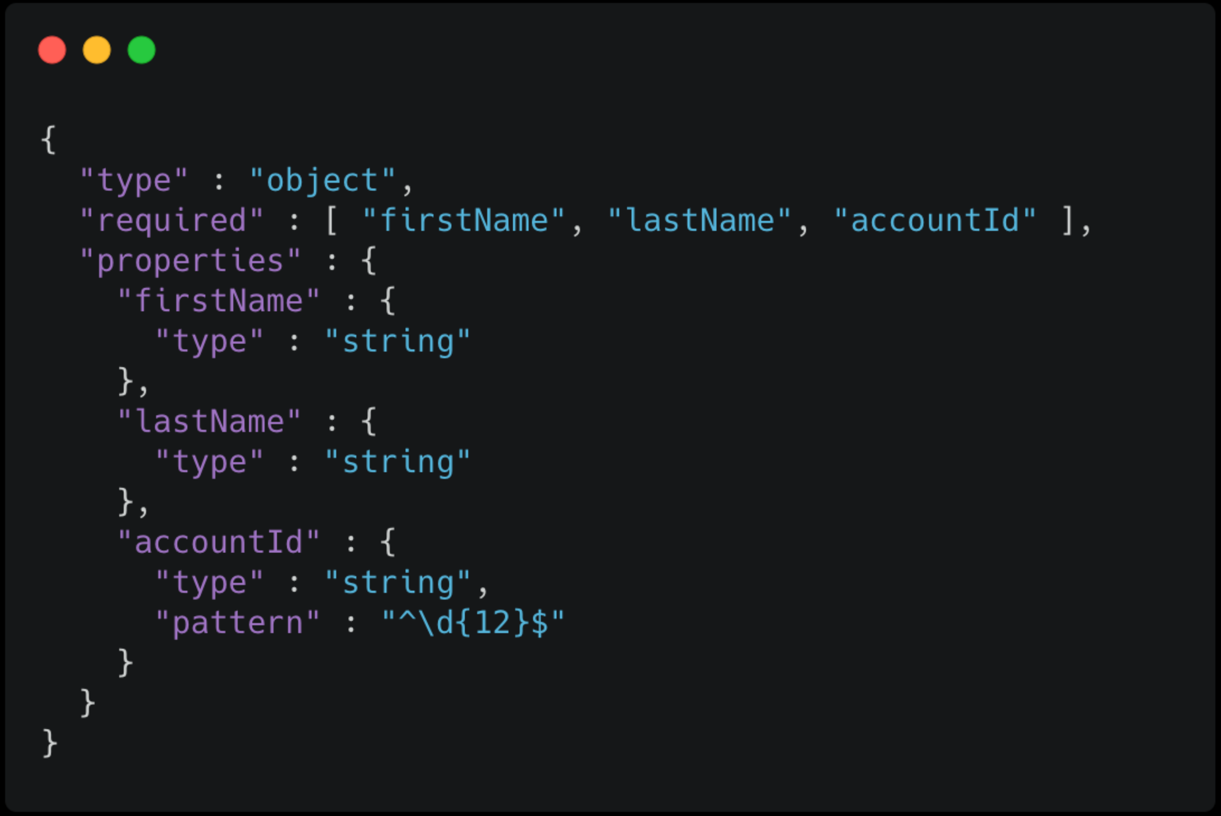
PERFORM DATA VALIDATION



Best practice 2: Data encryption and integrity

AMAZON API GATEWAY BASIC REQUEST VALIDATION

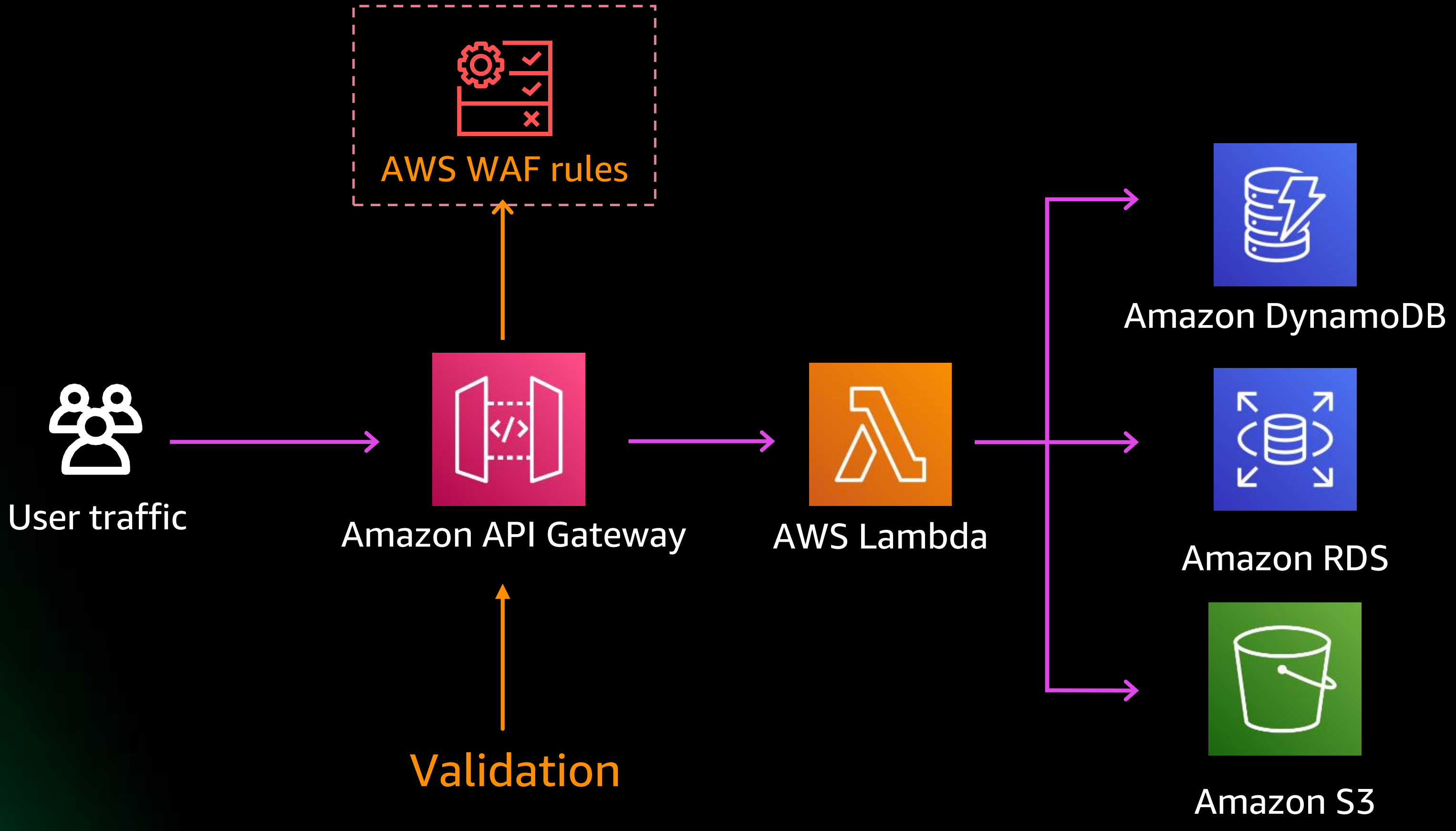
- Specify required parameters
- Specify model the payload must adhere to
- Assign to API method



```
{
  "type" : "object",
  "required" : [ "firstName", "lastName", "accountId" ],
  "properties" : {
    "firstName" : {
      "type" : "string"
    },
    "lastName" : {
      "type" : "string"
    },
    "accountId" : {
      "type" : "string",
      "pattern" : "^\d{12}$"
    }
  }
}
```

Best practice 2: Data encryption and integrity

PROTECT AGAINST COMMON WEB EXPLOITS

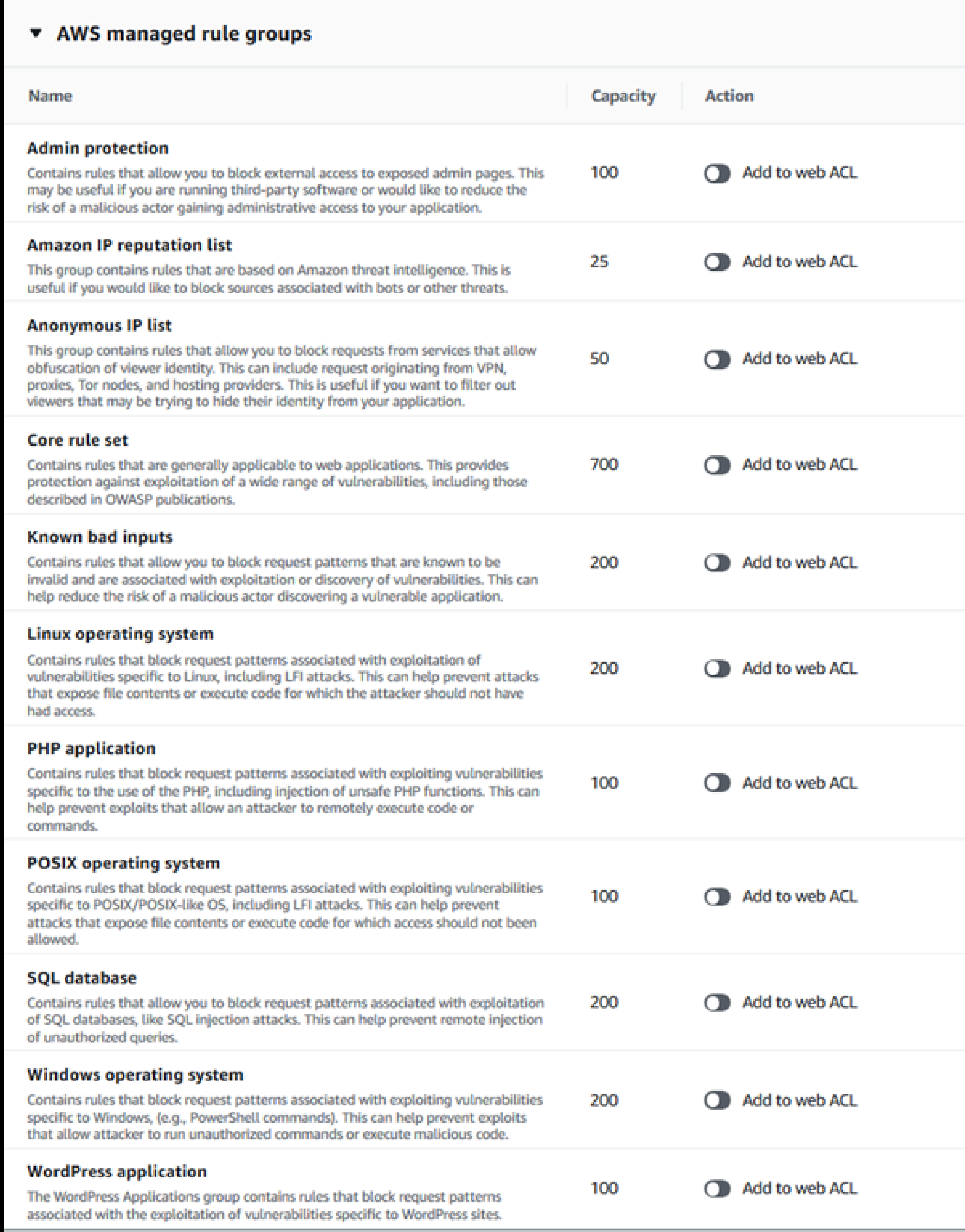


AWS Managed Rules within AWS WAF

Preconfigured rules

- Covers common attack vectors and threats
- Curated and maintained by SRT
- Influenced by OWASP Top 10

Note: Applicable for AWS WAF v2

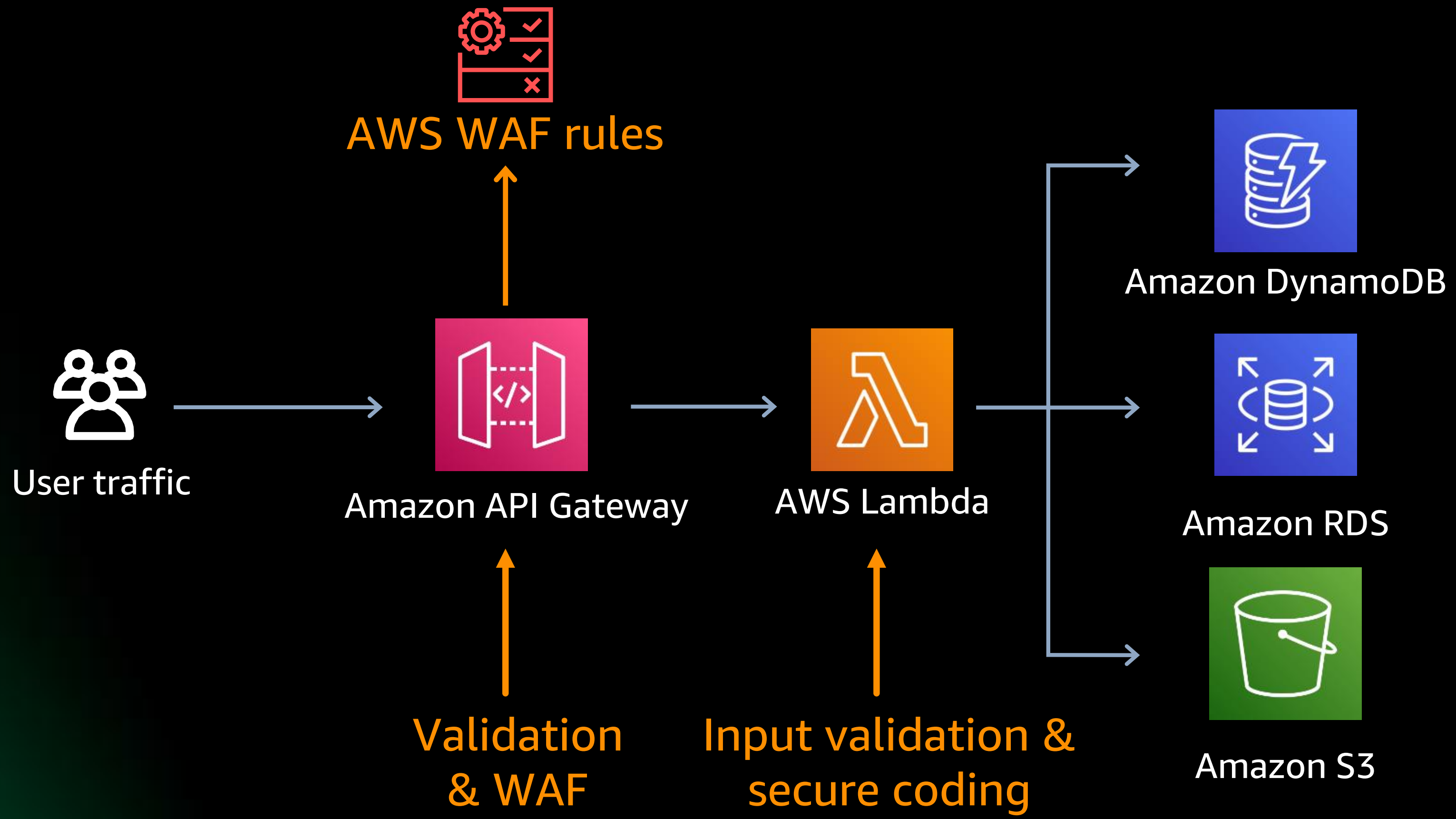


▼ AWS managed rule groups

Name	Capacity	Action
Admin protection Contains rules that allow you to block external access to exposed admin pages. This may be useful if you are running third-party software or would like to reduce the risk of a malicious actor gaining administrative access to your application.	100	○ Add to web ACL
Amazon IP reputation list This group contains rules that are based on Amazon threat intelligence. This is useful if you would like to block sources associated with bots or other threats.	25	○ Add to web ACL
Anonymous IP list This group contains rules that allow you to block requests from services that allow obfuscation of viewer identity. This can include request originating from VPN, proxies, Tor nodes, and hosting providers. This is useful if you want to filter out viewers that may be trying to hide their identity from your application.	50	○ Add to web ACL
Core rule set Contains rules that are generally applicable to web applications. This provides protection against exploitation of a wide range of vulnerabilities, including those described in OWASP publications.	700	○ Add to web ACL
Known bad inputs Contains rules that allow you to block request patterns that are known to be invalid and are associated with exploitation or discovery of vulnerabilities. This can help reduce the risk of a malicious actor discovering a vulnerable application.	200	○ Add to web ACL
Linux operating system Contains rules that block request patterns associated with exploitation of vulnerabilities specific to Linux, including LFI attacks. This can help prevent attacks that expose file contents or execute code for which the attacker should not have had access.	200	○ Add to web ACL
PHP application Contains rules that block request patterns associated with exploiting vulnerabilities specific to the use of the PHP, including injection of unsafe PHP functions. This can help prevent exploits that allow an attacker to remotely execute code or commands.	100	○ Add to web ACL
POSIX operating system Contains rules that block request patterns associated with exploiting vulnerabilities specific to POSIX/POSIX-like OS, including LFI attacks. This can help prevent attacks that expose file contents or execute code for which access should not been allowed.	100	○ Add to web ACL
SQL database Contains rules that allow you to block request patterns associated with exploitation of SQL databases, like SQL injection attacks. This can help prevent remote injection of unauthorized queries.	200	○ Add to web ACL
Windows operating system Contains rules that block request patterns associated with exploiting vulnerabilities specific to Windows, (e.g., PowerShell commands). This can help prevent exploits that allow attacker to run unauthorized commands or execute malicious code.	200	○ Add to web ACL
WordPress application The WordPress Applications group contains rules that block request patterns associated with the exploitation of vulnerabilities specific to WordPress sites.	100	○ Add to web ACL

Best practice 2: Data encryption and integrity

PROTECT DATA IN TRANSIT

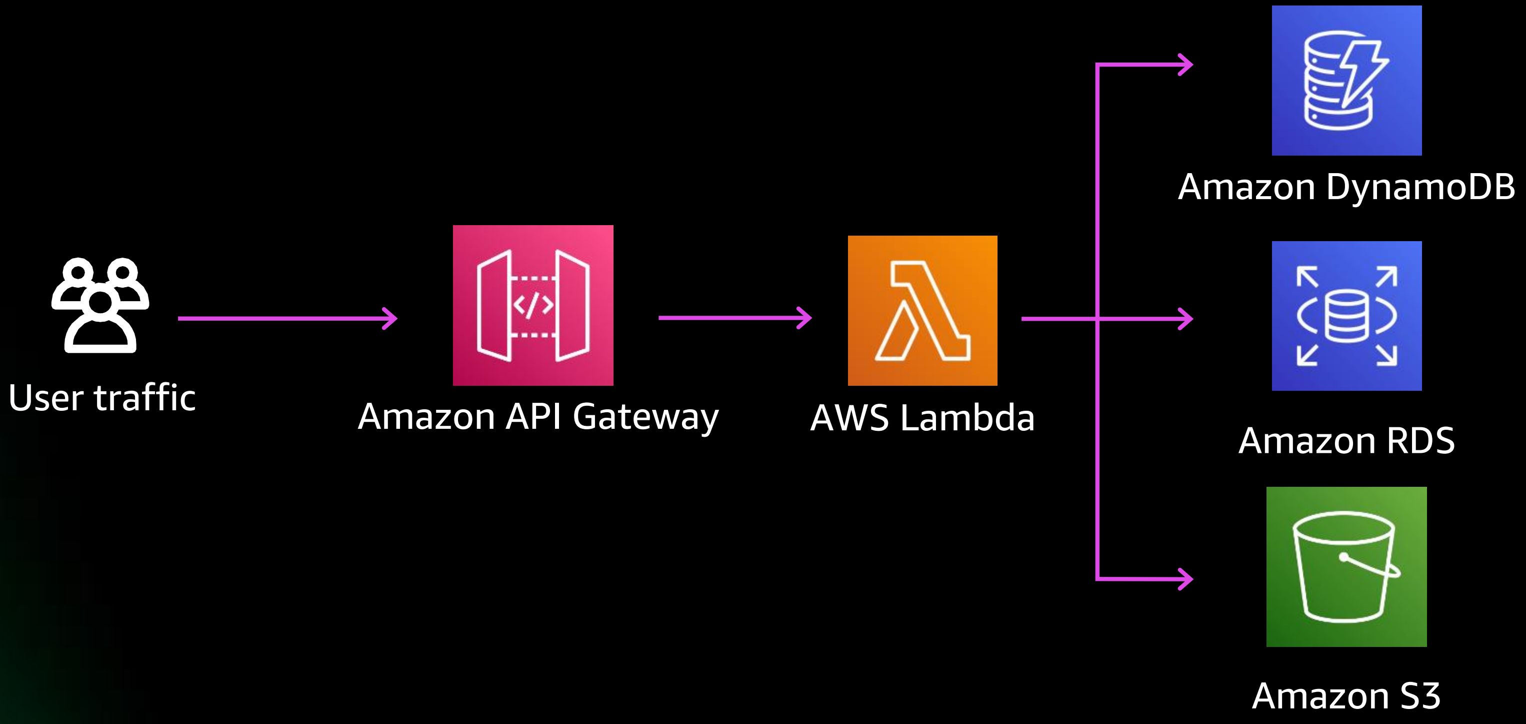


Best practice 2: Data encryption and integrity

SECURE CODING BEST PRACTICES

- Perform input validation before processing data
  - Validate the events, data types
  - Only process expected input
- Safe deserialization
  - OWASP Safe Deserialization Cheat Sheet: s12d.com/owasp-deserialization
- Check for vulnerabilities on your dependencies
  - OWASP Dependency Check: s12d.com/owasp-dep-check
  - Third-party tools
- Remove unused dependencies
  - depcheck: s12d.com/depcheck

Best practice 3: Monitoring, logging, and configuration management



Best practice 3: What to consider

MONITORING, LOGGING, AND CONFIGURATION MANAGEMENT

- Use monitoring tools to identify and report unwanted behavior such as
  - Wrong credentials
  - Unauthorized access to resources
  - Excessive invocation of functions
  - Unusually long running time
- Ensure sufficient logging is enabled for all components
  - Avoid logging sensitive data
- Perform regular auditing of configuration

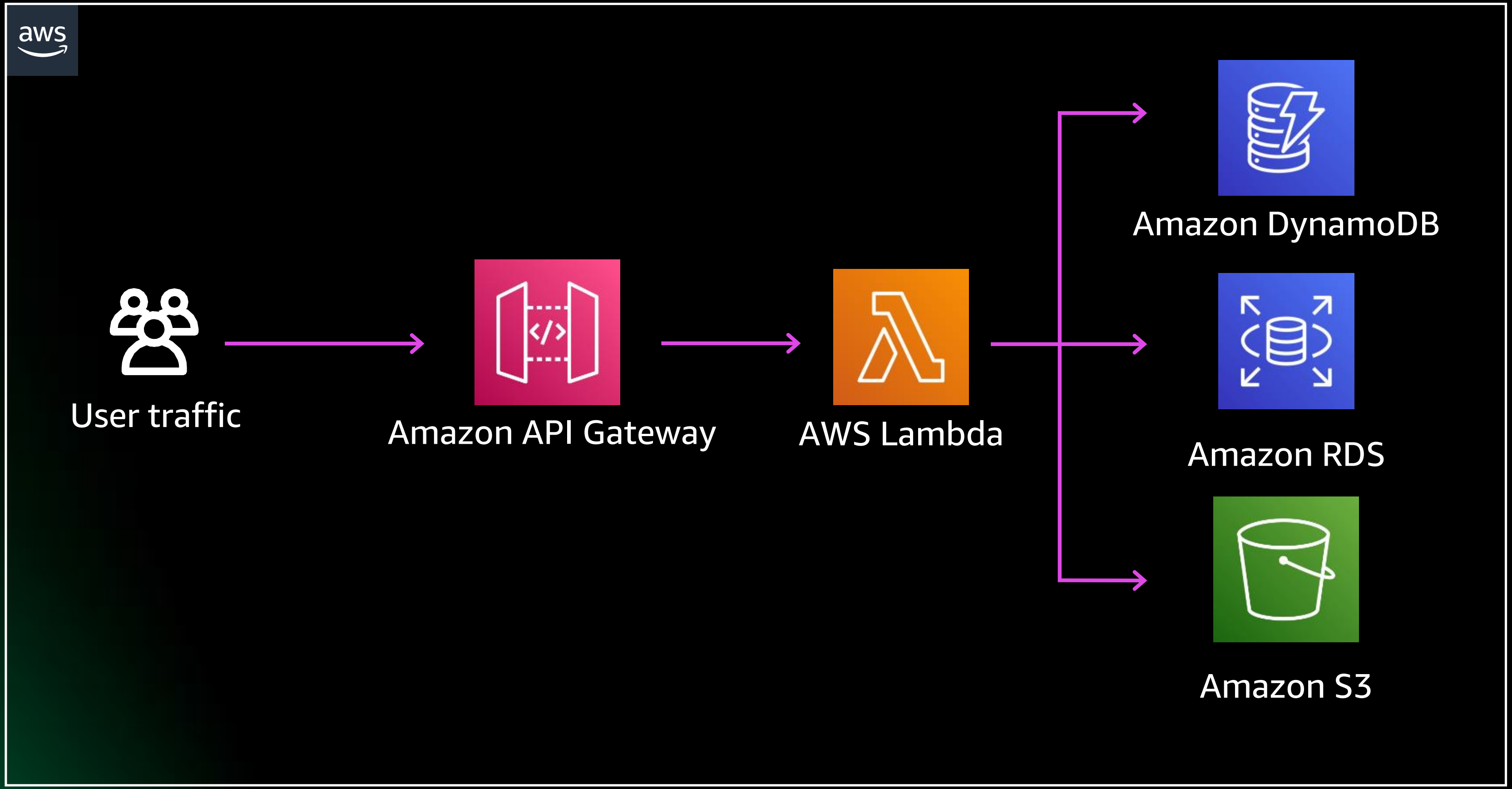
OWASP Serverless Top 10
S6:2017 Security Misconfiguration
S10:2017 Insufficient Logging and Monitoring

AWS Well-Architected Framework
Enable traceability
Apply security at all layers

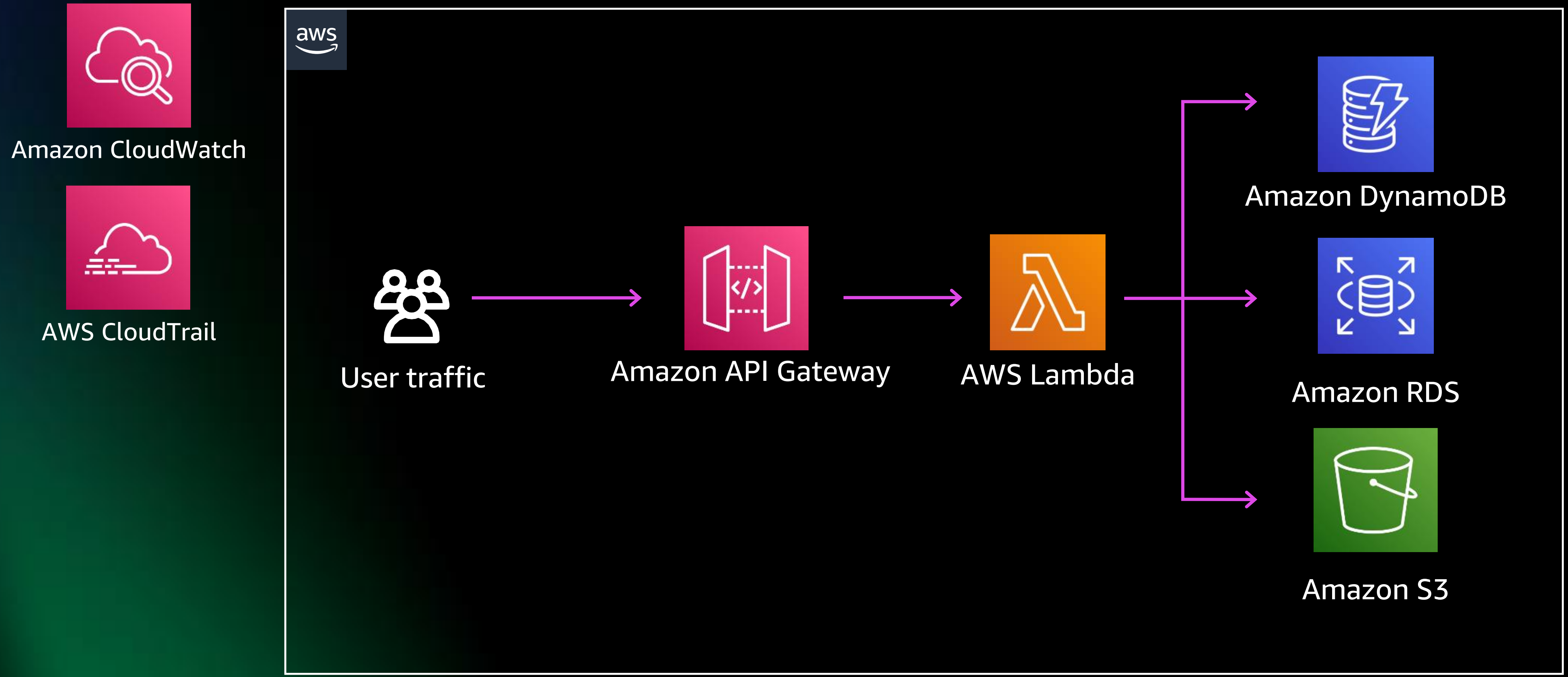
Best practice 3: Monitoring, logging, and configuration management

Amazon CloudWatch

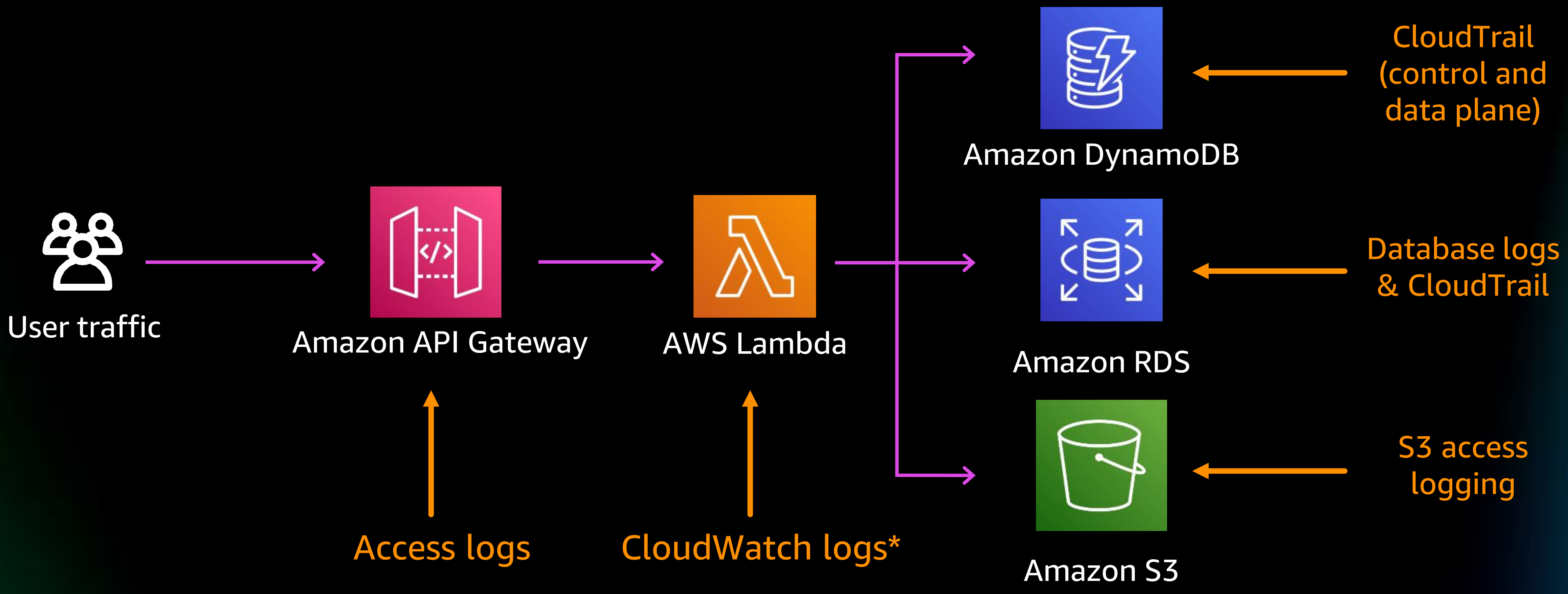
- Metrics
- Lambda Insights
- Alarms
- Dashboards
- Logging



Best practice 3: Monitoring, logging, and configuration management



Best practice 3: Monitoring, logging, and configuration management



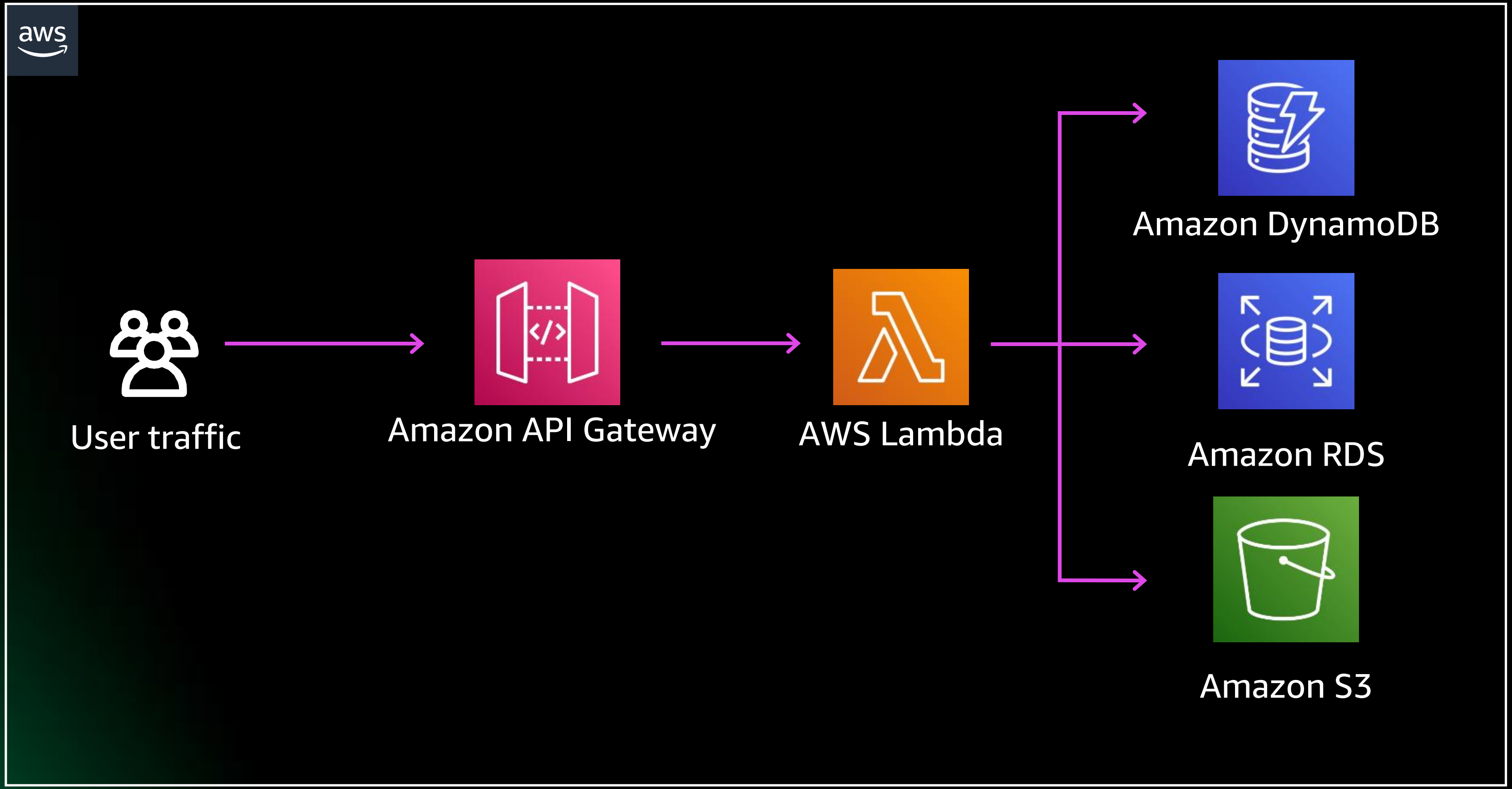
*Help ensure Lambda function execution role has permissions to CloudWatch

Best practice 3: Monitoring, logging, and configuration management

Amazon CloudWatch

AWS CloudTrail

AWS Config



User traffic

Amazon API Gateway

AWS Lambda

Amazon DynamoDB

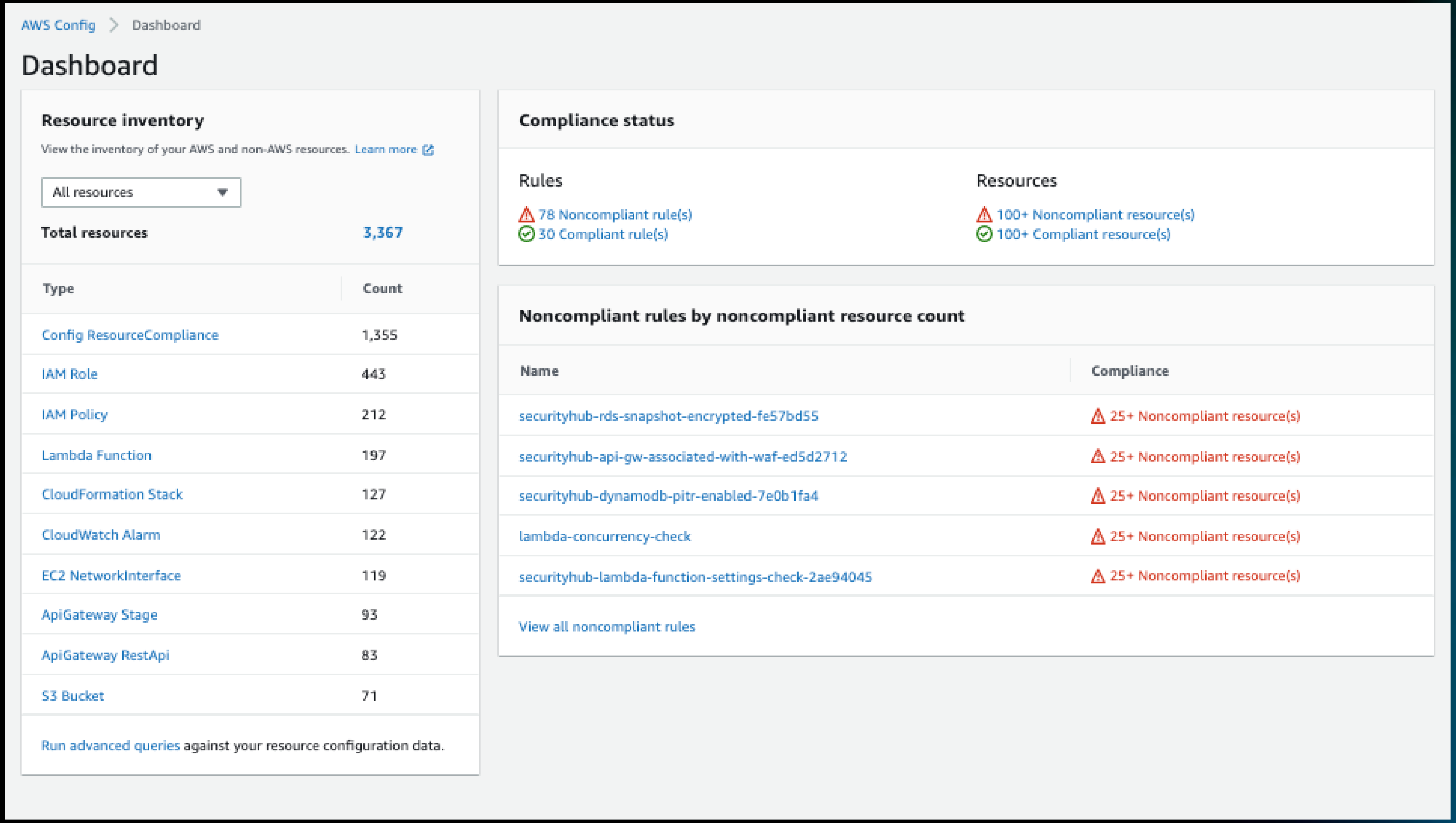
Amazon RDS

Amazon S3

Best practice 3: Monitoring, logging, and configuration management

AWS Config

- Configuration **auditor**
- Monitors **configuration changes over time**
- Evaluates the configuration against **policies** defined using **AWS Config rules**
- **Alerts** you if the configuration is noncompliant with your policies



AWS Config > Dashboard

Dashboard

Resource inventory

View the inventory of your AWS and non-AWS resources. Learn more

All resources ▼

Total resources 3,367

Type	Count
Config ResourceCompliance	1,355
IAM Role	443
IAM Policy	212
Lambda Function	197
CloudFormation Stack	127
CloudWatch Alarm	122
EC2 NetworkInterface	119
ApiGateway Stage	93
ApiGateway RestApi	83
S3 Bucket	71

Run advanced queries against your resource configuration data.

Compliance status

Rules

⚠ 78 Noncompliant rule(s)
✓ 30 Compliant rule(s)

Resources

⚠ 100+ Noncompliant resource(s)
✓ 100+ Compliant resource(s)

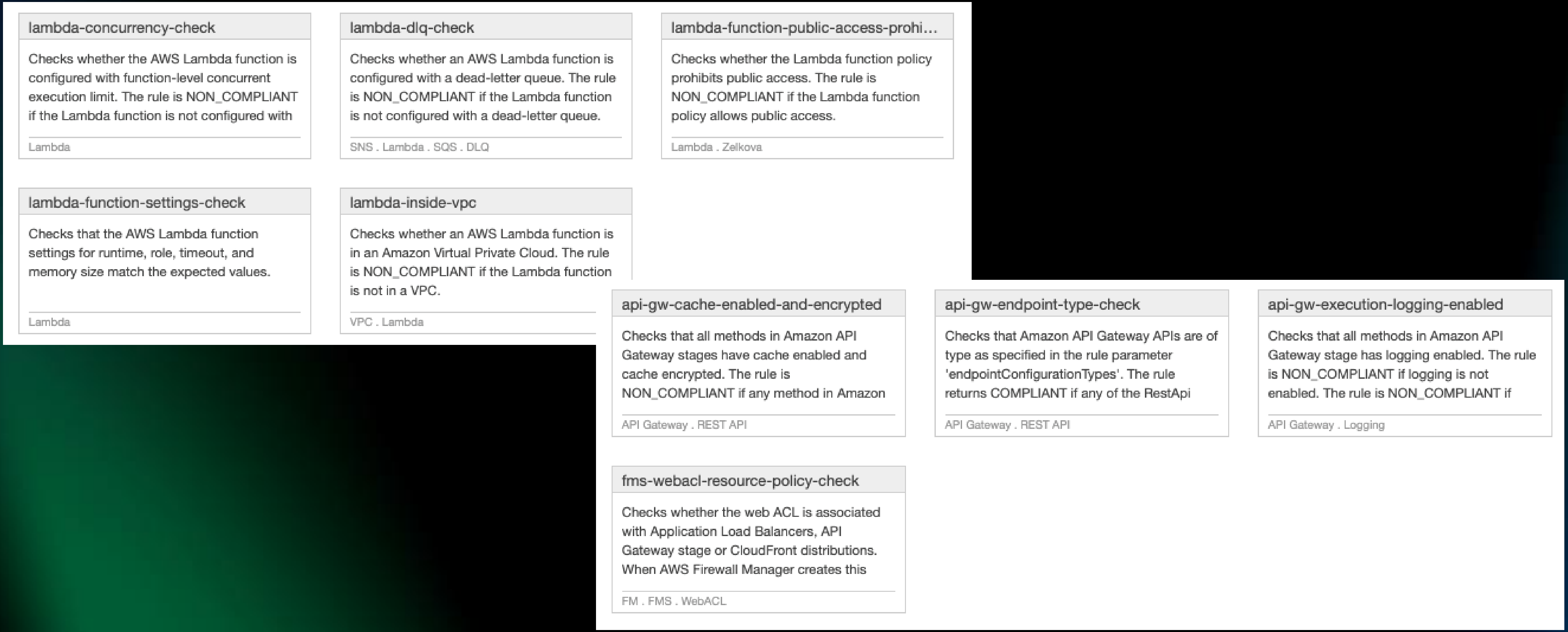
Noncompliant rules by noncompliant resource count

Name	Compliance
securityhub-rds-snapshot-encrypted-fe57bd55	⚠ 25+ Noncompliant resource(s)
securityhub-api-gw-associated-with-waf-ed5d2712	⚠ 25+ Noncompliant resource(s)
securityhub-dynamodb-pitr-enabled-7e0b1fa4	⚠ 25+ Noncompliant resource(s)
lambda-concurrency-check	⚠ 25+ Noncompliant resource(s)
securityhub-lambda-function-settings-check-2ae94045	⚠ 25+ Noncompliant resource(s)

View all noncompliant rules

Best practice 3: Monitoring, logging, and configuration management

EXAMPLE MANAGED AWS CONFIG RULES



lambda-concurrency-check

Checks whether the AWS Lambda function is configured with function-level concurrent execution limit. The rule is NON_COMPLIANT if the Lambda function is not configured with

Lambda

lambda-dlq-check

Checks whether an AWS Lambda function is configured with a dead-letter queue. The rule is NON_COMPLIANT if the Lambda function is not configured with a dead-letter queue.

SNS . Lambda . SQS . DLQ

lambda-function-public-access-prohi...

Checks whether the Lambda function policy prohibits public access. The rule is NON_COMPLIANT if the Lambda function policy allows public access.

Lambda . Zelkova

lambda-function-settings-check

Checks that the AWS Lambda function settings for runtime, role, timeout, and memory size match the expected values.

Lambda

lambda-inside-vpc

Checks whether an AWS Lambda function is in an Amazon Virtual Private Cloud. The rule is NON_COMPLIANT if the Lambda function is not in a VPC.

VPC . Lambda

api-gw-cache-enabled-and-encrypted

Checks that all methods in Amazon API Gateway stages have cache enabled and cache encrypted. The rule is NON_COMPLIANT if any method in Amazon

API Gateway . REST API

api-gw-endpoint-type-check

Checks that Amazon API Gateway APIs are of type as specified in the rule parameter 'endpointConfigurationTypes'. The rule returns COMPLIANT if any of the RestApi

API Gateway . REST API

api-gw-execution-logging-enabled

Checks that all methods in Amazon API Gateway stage has logging enabled. The rule is NON_COMPLIANT if logging is not enabled. The rule is NON_COMPLIANT if

API Gateway . Logging

fms-webacl-resource-policy-check

Checks whether the web ACL is associated with Application Load Balancers, API Gateway stage or CloudFront distributions. When AWS Firewall Manager creates this

FM . FMS . WebACL

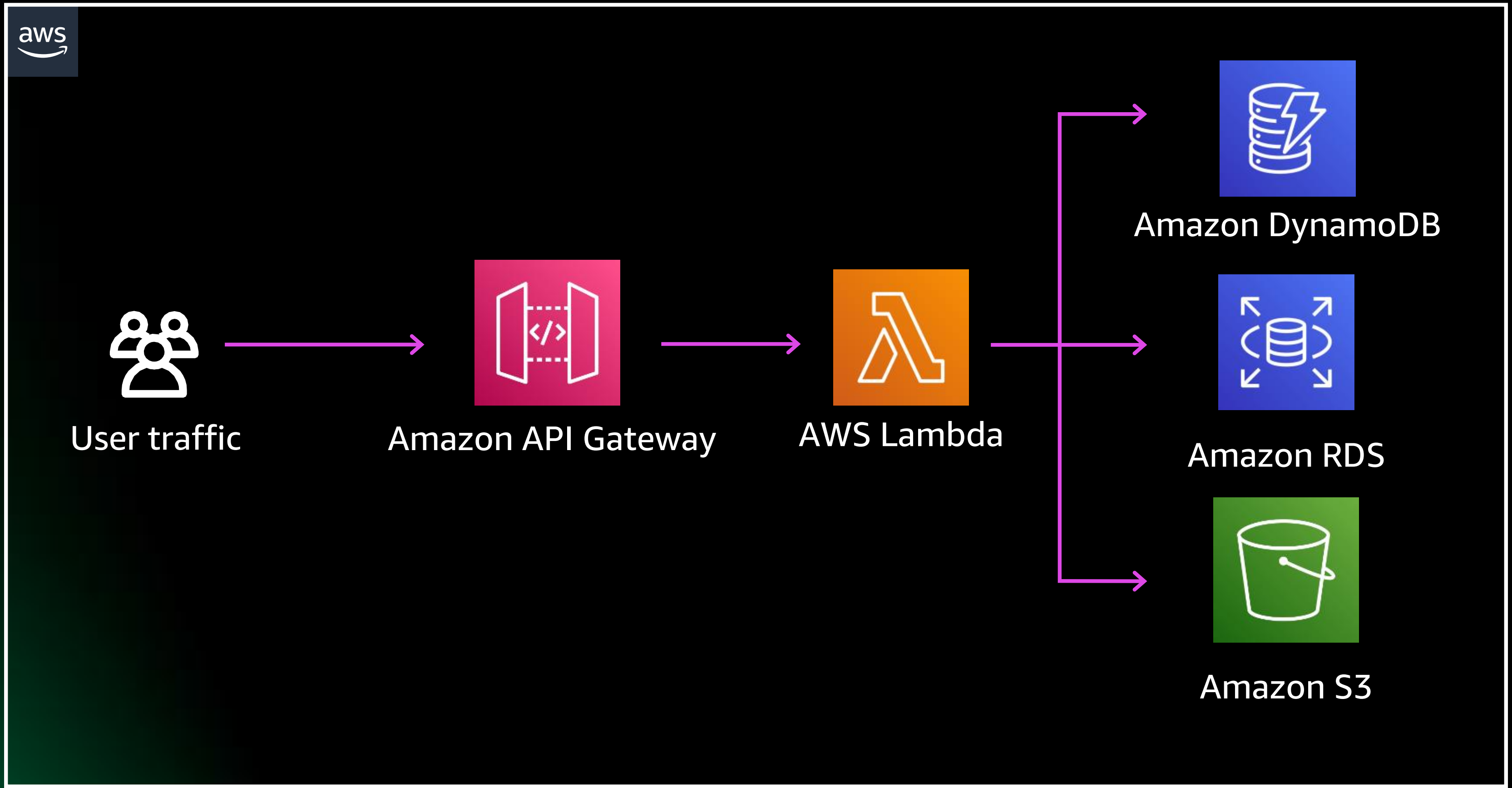
Best practice 3: Monitoring, logging, and configuration management

Amazon CloudWatch

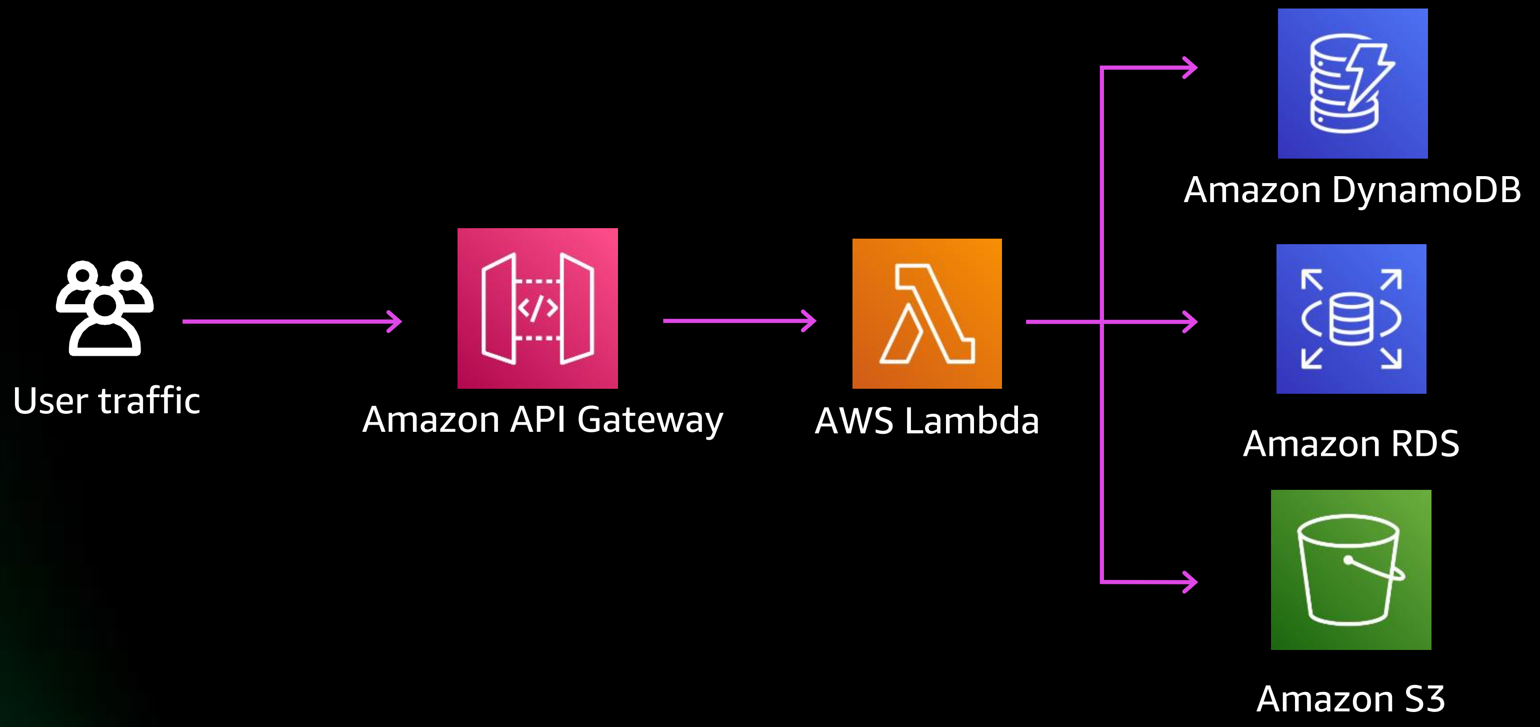
AWS CloudTrail

AWS Config

AWS X-Ray



Best practice 4: Denial of service

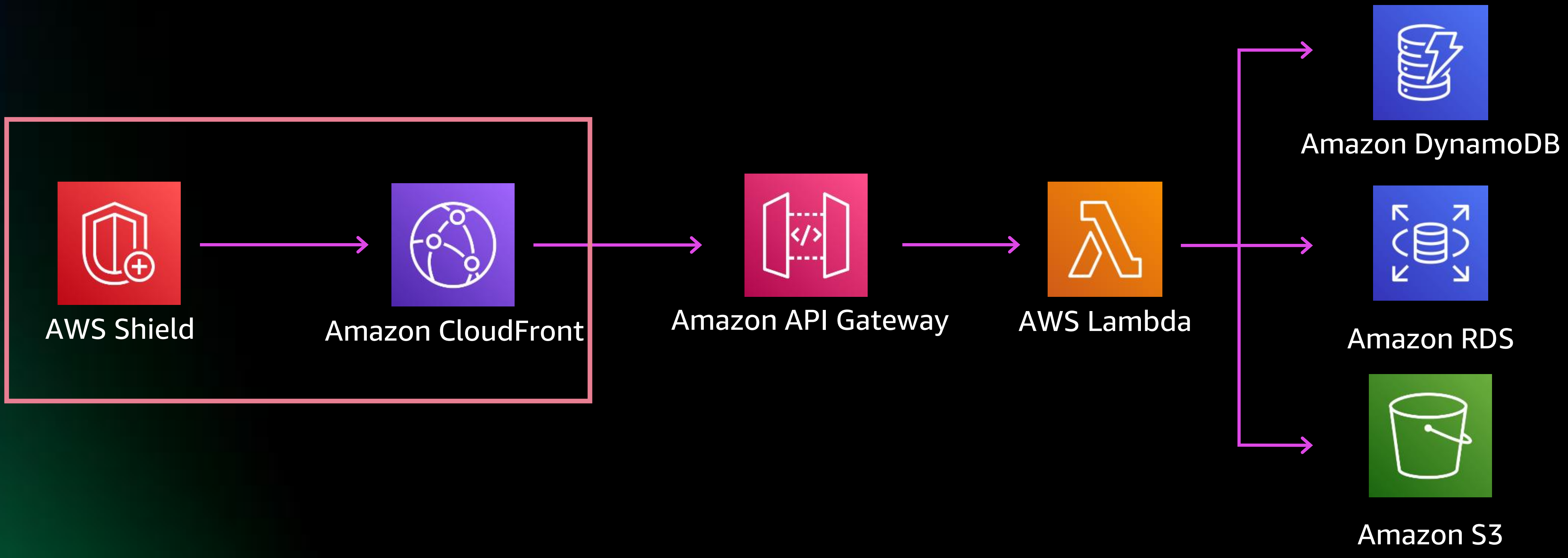


Best practice 4: What to consider

DENIAL OF SERVICE AND INFRASTRUCTURE PROTECTION

- DDoS protection
- Throttling/rate limiting
- Network boundaries

Best practice 4: Denial of service



Best practice 4: Denial of service

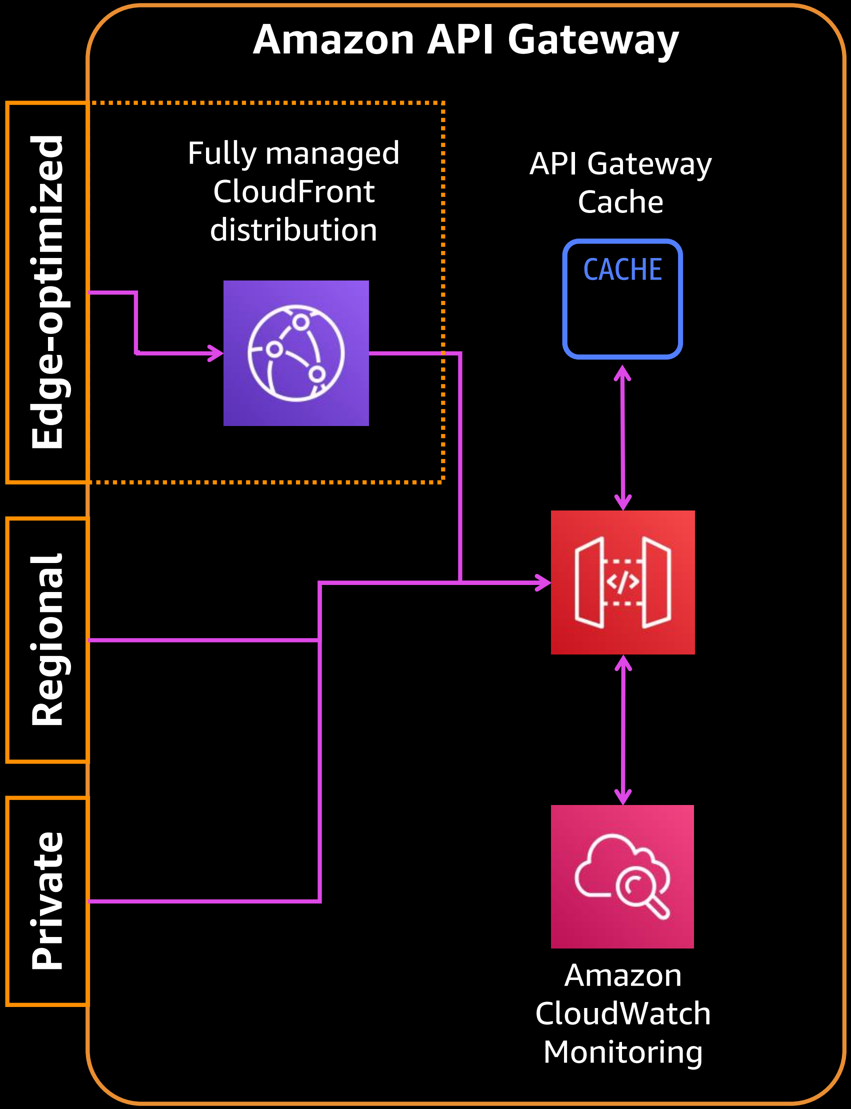
AMAZON API GATEWAY ENDPOINTS

Edge-optimized

- Utilizes CloudFront to reduce TLS connection overhead (reduces roundtrip time)
- Designed for a globally distributed set of clients

Private

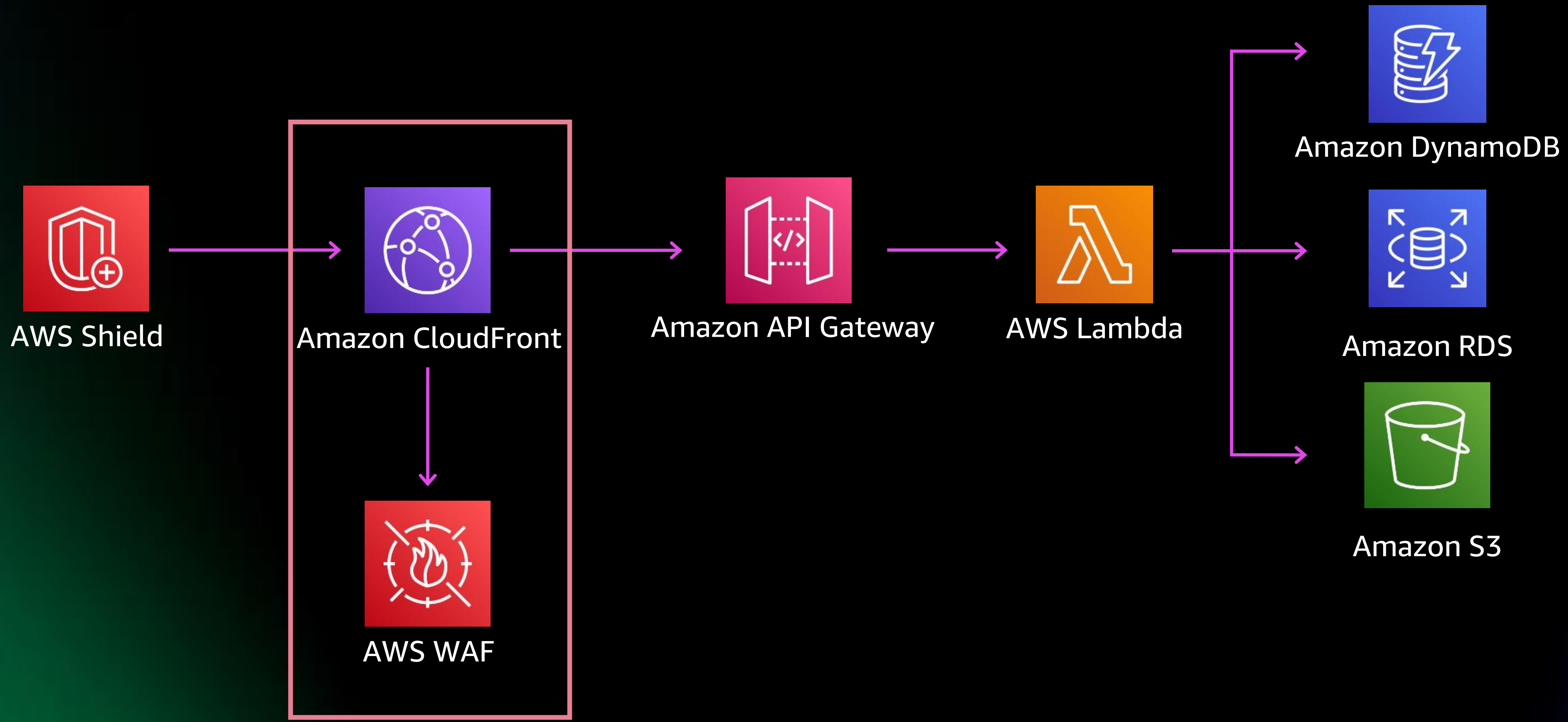
- Only accessible from within VPC (and networks connected to VPC)
- Designed for building APIs used internally or by private microservices



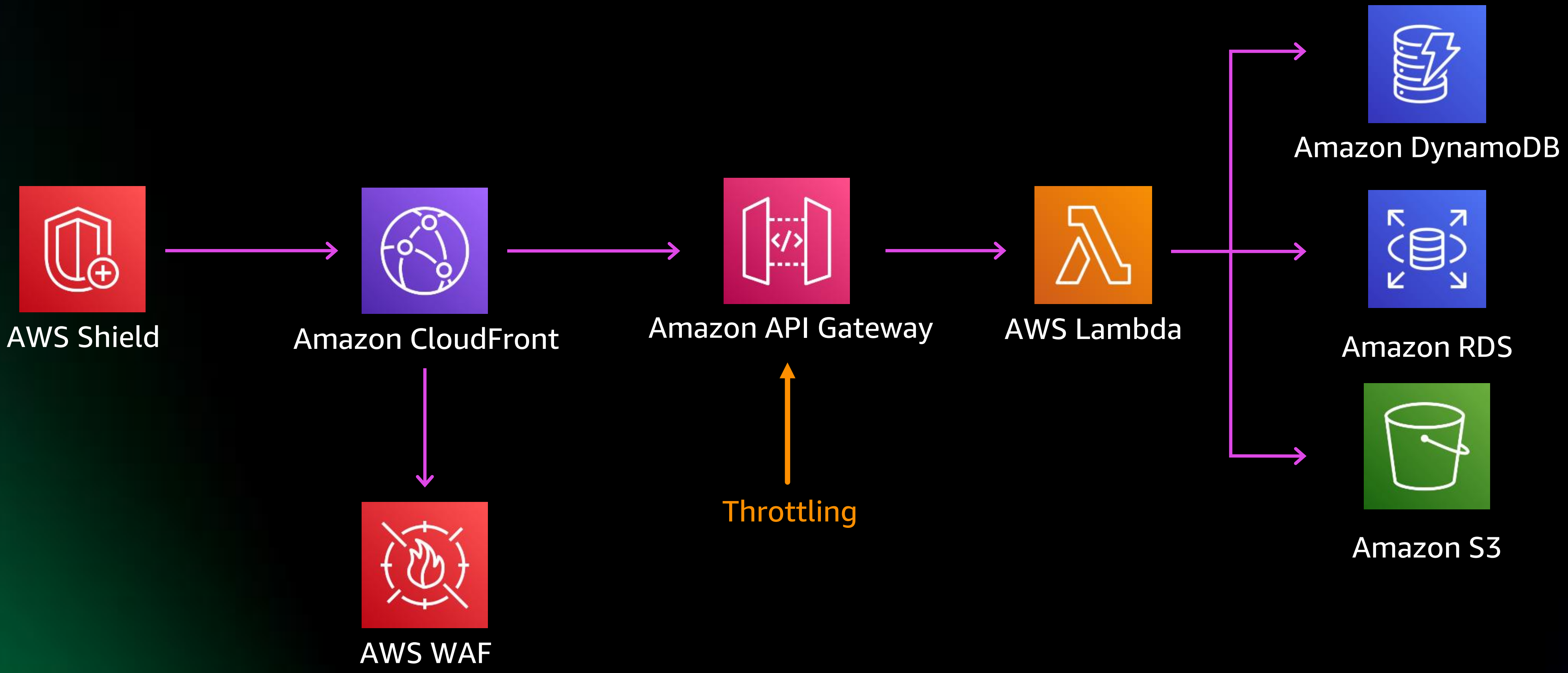
Regional

- Recommended API type for general use cases
- Designed for building APIs for clients in the same Region

Best practice 4: Denial of service

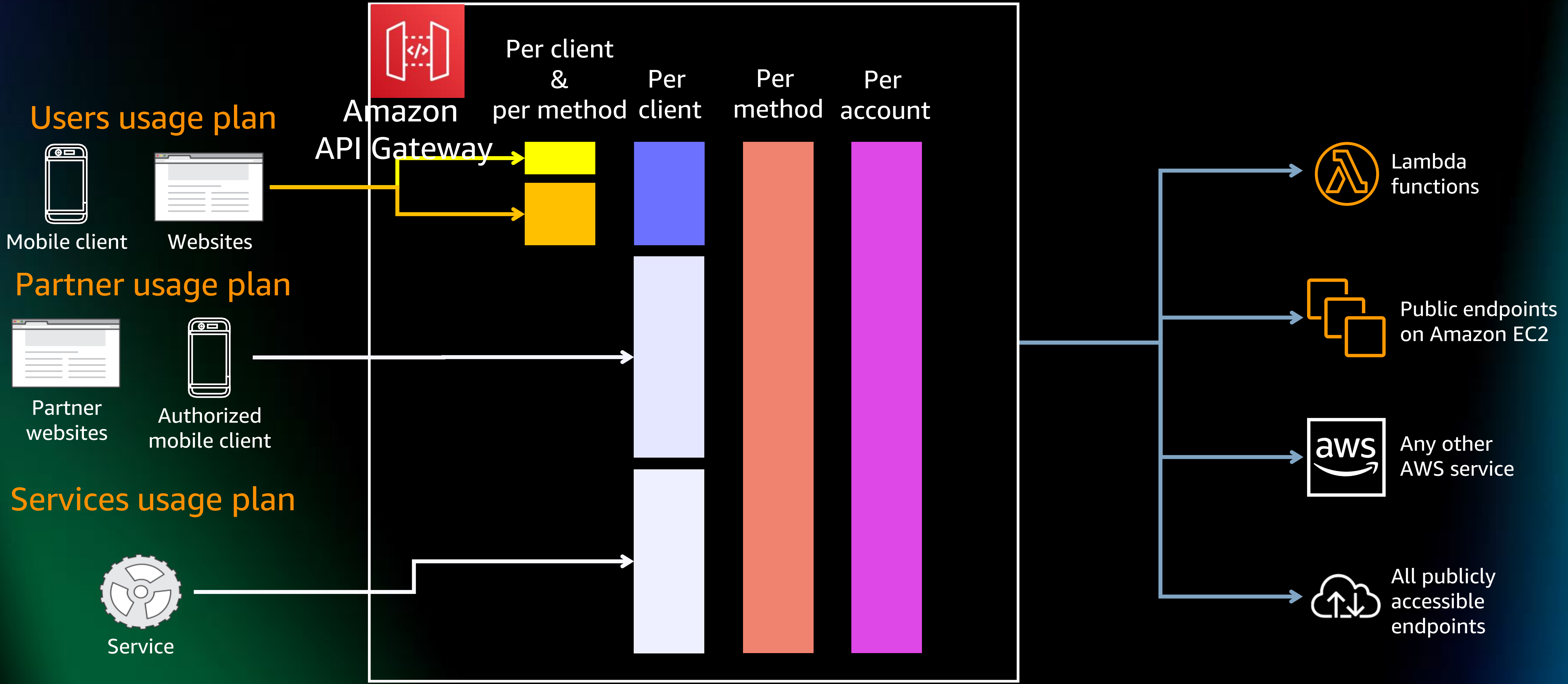


Best practice 4: Denial of service



Best practice 4: Denial of service

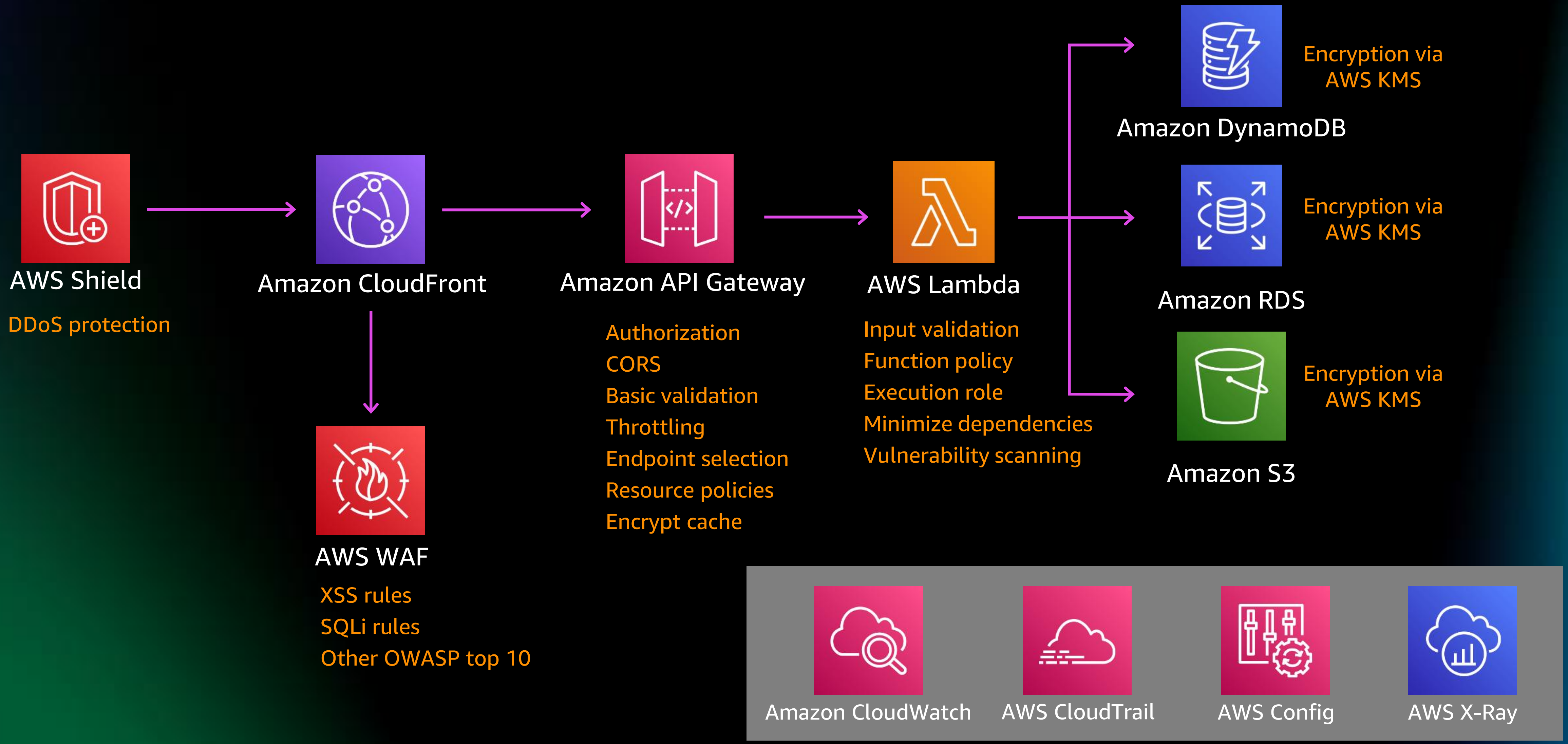
IMPLEMENT THROTTLING



Agenda

- Overview of serverless security
- Mental model for serverless security
- Best practices for serverless applications
- Recap!

Recap



AWS Shield

DDoS protection

Amazon CloudFront

AWS WAF

- XSS rules
- SQLi rules
- Other OWASP top 10

Amazon API Gateway

- Authorization
- CORS
- Basic validation
- Throttling
- Endpoint selection
- Resource policies
- Encrypt cache

AWS Lambda

- Input validation
- Function policy
- Execution role
- Minimize dependencies
- Vulnerability scanning

Amazon DynamoDB

Encryption via AWS KMS

Amazon RDS

Encryption via AWS KMS

Amazon S3

Encryption via AWS KMS

Amazon CloudWatch

AWS CloudTrail

AWS Config

AWS X-Ray

Key Takeaways

Serverless security is

- Balanced toward the application, not the infrastructure
- More fine-grained
- Not to be taken for granted

Learn more about other security solutions:

https://aws.amazon.com/security/

Additional resources

OWASP Serverless Top 10: https://bit.ly/3LjYcyl

Serverless Applications Lens – AWS Well-Architected Framework Security Pillar: https://go.aws/3B6YB2b

Serverless samples: https://bit.ly/3RRg31s

Security Overview of AWS Lambda

https://go.aws/3QAekMT

Security Overview of Amazon API Gateway

https://bit.ly/3BaeMvE

Visit the Modern Applications resource hub

Dive deeper with these resources to help you develop an effective plan for your modernization journey.

- Build modern applications on AWS
- Business value of cloud modernization
- An introduction to event-driven architectures
- Accelerate full-stack web and mobile app development
- Determining the total cost of ownership: Comparing serverless and server-based technologies
- Building event-driven architectures with AWS
- Continuous learning, continuous modernization

Visit resource hub

https://tinyurl.com/modern-apps-aws

AWS Training and Certification

Get started with Free Digital Training for you and your team today

Achieve key milestones and plan your next steps with the AWS Modern Application skills training

Access 500+ free digital courses with AWS Skill Builder

Earn an industry-recognized credential:
AWS Certified Developer – Associate
AWS Certified DevOps – Professional

Create a self-paced learning roadmap
AWS ramp-up guide - Developer
AWS ramp-up guide - DevOps

Thank you!

Kapil Gambhir

kgambhi@amazon.com

Thank you for attending AWS Innovate Modern Applications Edition

We hope you found it interesting! A kind reminder to **complete the survey.** Let us know what you thought of today's event and how we can improve the event experience for you in the future.

- aws-apj-marketing@amazon.com
- twitter.com/AWSCloud
- facebook.com/AmazonWebServices
- youtube.com/user/AmazonWebServices
- slideshare.net/AmazonWebServices
- twitch.tv/aws