
AWS
re:Invent

SVS307

Best practices for securing your serverless applications

Rob Sutter

Sr. Developer Advocate

AWS

AWS re:Invent

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Agenda

- How is serverless application security different?
- Similarities to traditional application security
- Service-specific security guidance
- Considering trade-offs

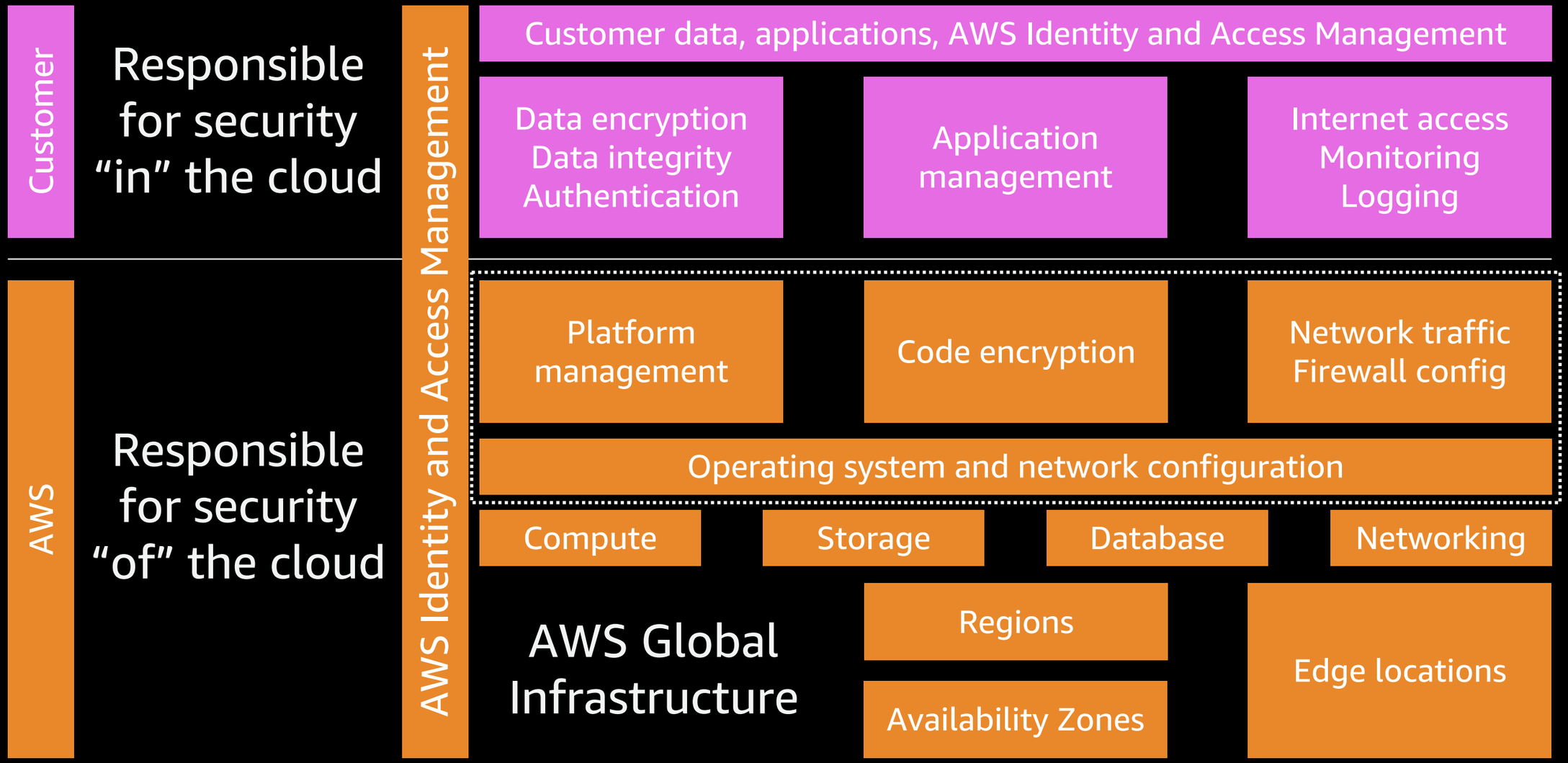


© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Differences

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

AWS shared responsibility model



AWS shared responsibility model

AWS assumes responsibility for these components of serverless applications



Finer-grained control gives you better security

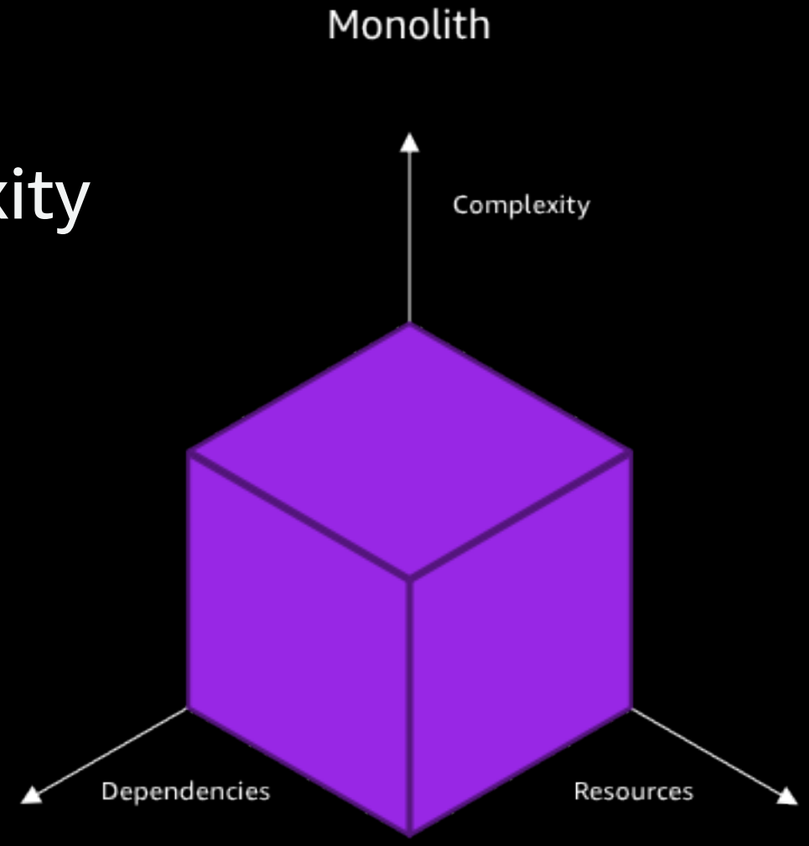
In a traditional application, every line of code is exposed to every vulnerability in every dependency and has access to every resource

Attack surface area = $\Sigma(c_f) * \Sigma(d_f)$ where:

- c_f = each function's computational complexity
- d_f = each function's dependencies

Potential impact = a * r where:

- a = attack surface area (see above)
- r = total number of accessible resources

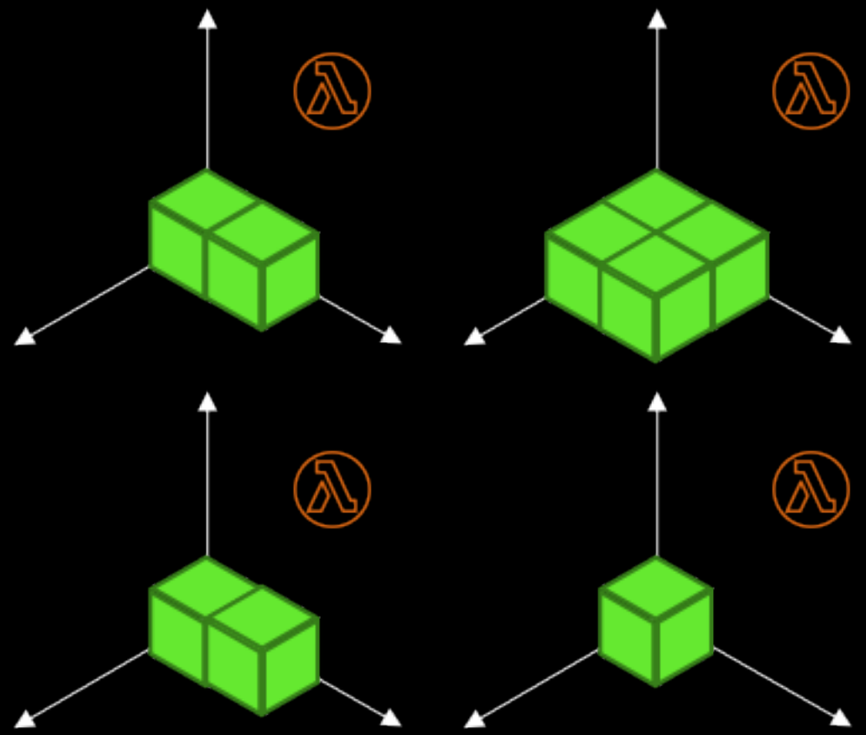


Finer-grained control gives you better security

In a well-architected serverless application, each unit of code is exposed only to the vulnerabilities in its specific logic and dependencies, and has access only to its own resources

Potential impact = $\Sigma(c_f * d_f * r_f)$ where:

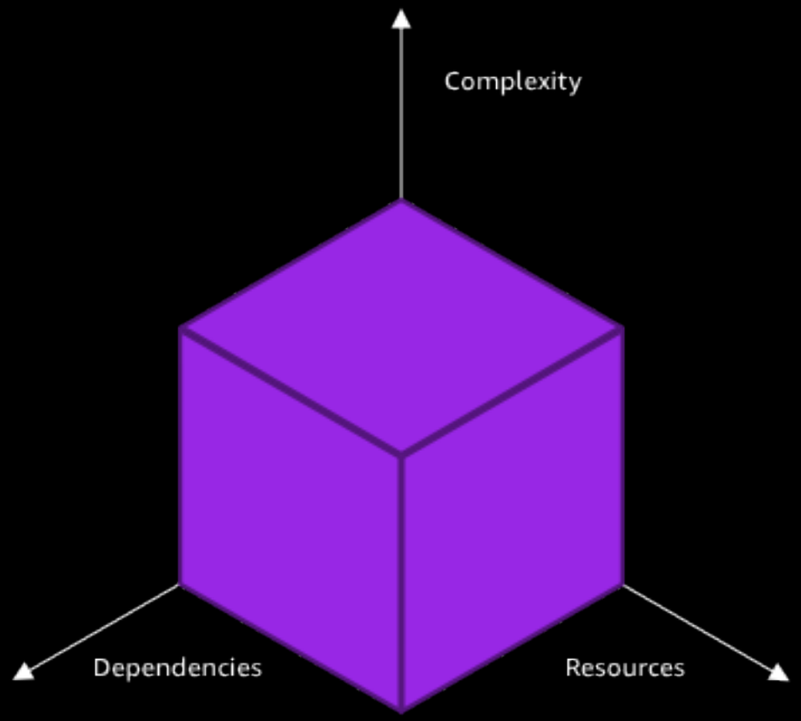
- c_f = each function's complexity
- d_f = each function's dependencies
- r_f = each function's resources



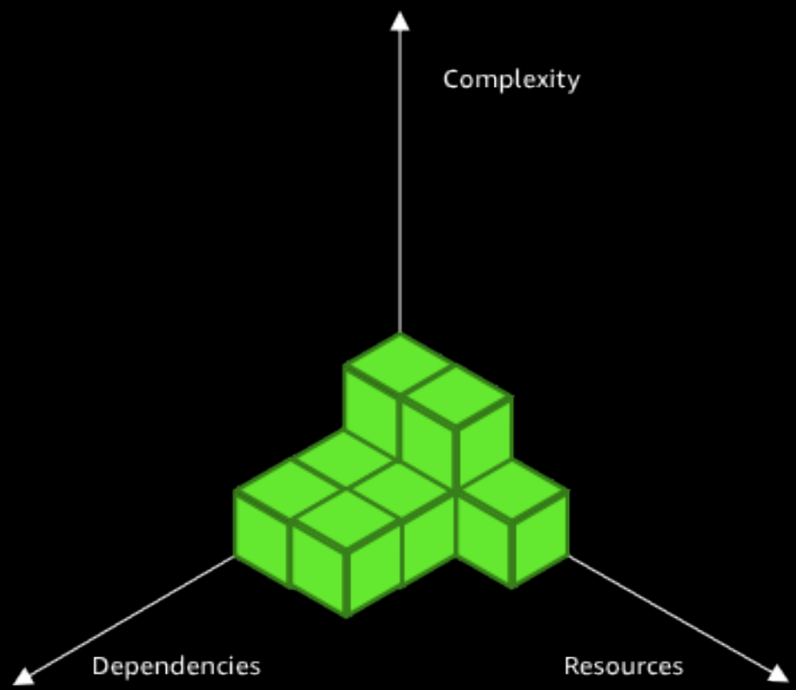
Finer-grained control gives you better security

In plain language, the potential security risk of a serverless application is lower, but it’s still present!

Monolith



Serverless application



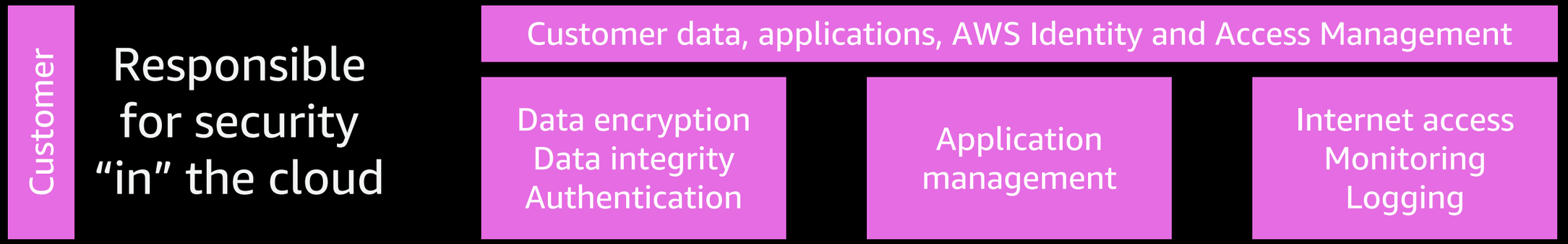
Similarities

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Serverless application security similarities

Security is not “free” with serverless. It still takes work!

- Application layer security
- Authentication and authorization
- Data encryption and integrity
- Monitoring and logging



OWASP Serverless Top Ten

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

s12d.com/owasp-top10

Application layer security

- Applications have different use cases and risk tolerances
- AWS enables customers to build according to their needs
- A security vulnerability in one application can be indistinguishable from a critical feature in another
    - Example: a business-to-consumer (B2C) platform startup enables cross-origin resource sharing (CORS) globally, whereas a financial institution restricts it entirely

Authentication and authorization

- Use available tooling
- Amazon offers Amazon Cognito
- Partners such as Auth0
- Don’t write your own!
- AWS Identity and Access Management (IAM) ties all the pieces together



AWS Identity and Access Management

Data encryption and integrity

- Identify and classify sensitive data
- Minimize storage of sensitive data to only what is necessary
- Protect data at rest
- Use infrastructure provider services for key management and encryption of stored data, secrets, and environment variables



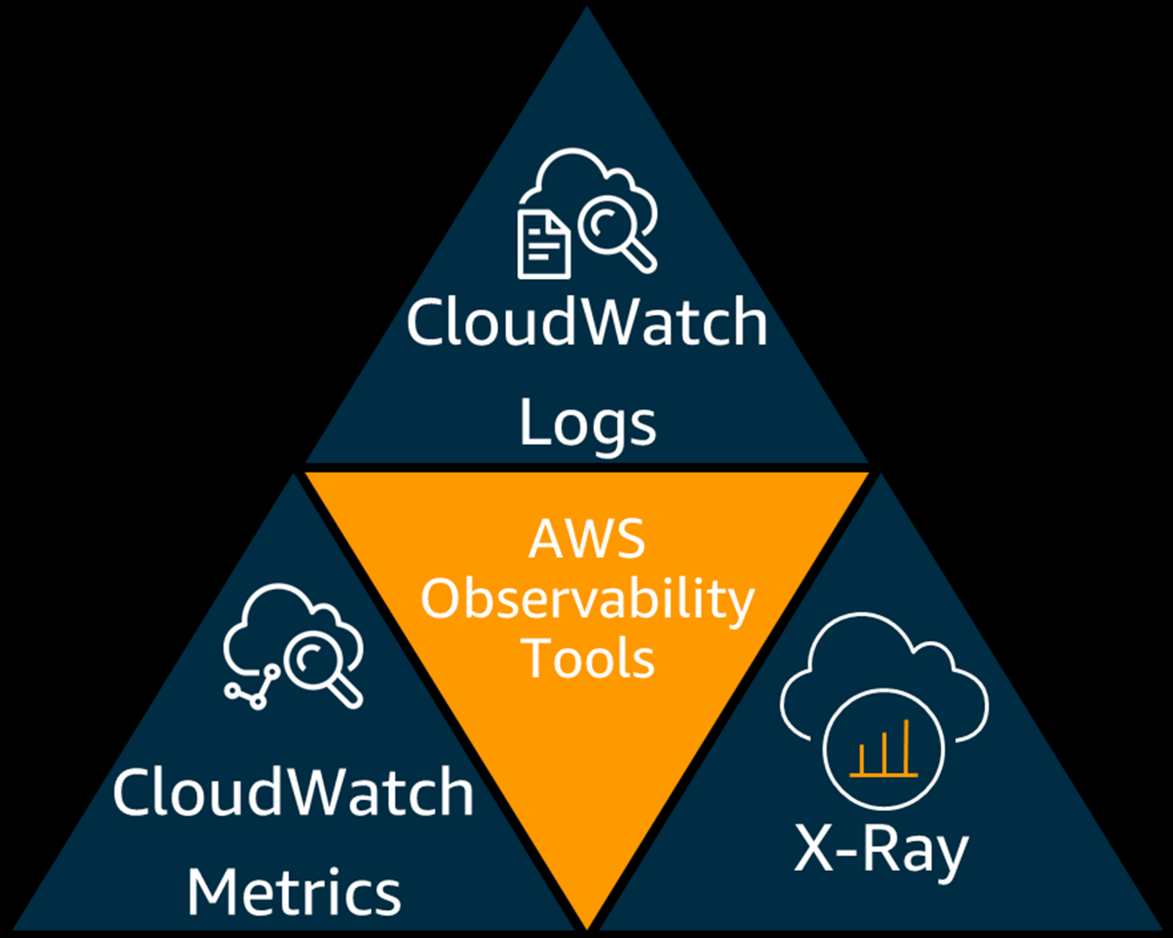
AWS Key Management Service



AWS Secrets Manager

Monitoring and logging

- Use monitoring tools provided by the service provider to identify and report unwanted behavior
  - Wrong credentials
  - Unauthorized access to resources
  - Excessive invocation of functions
  - Unusually long runtime



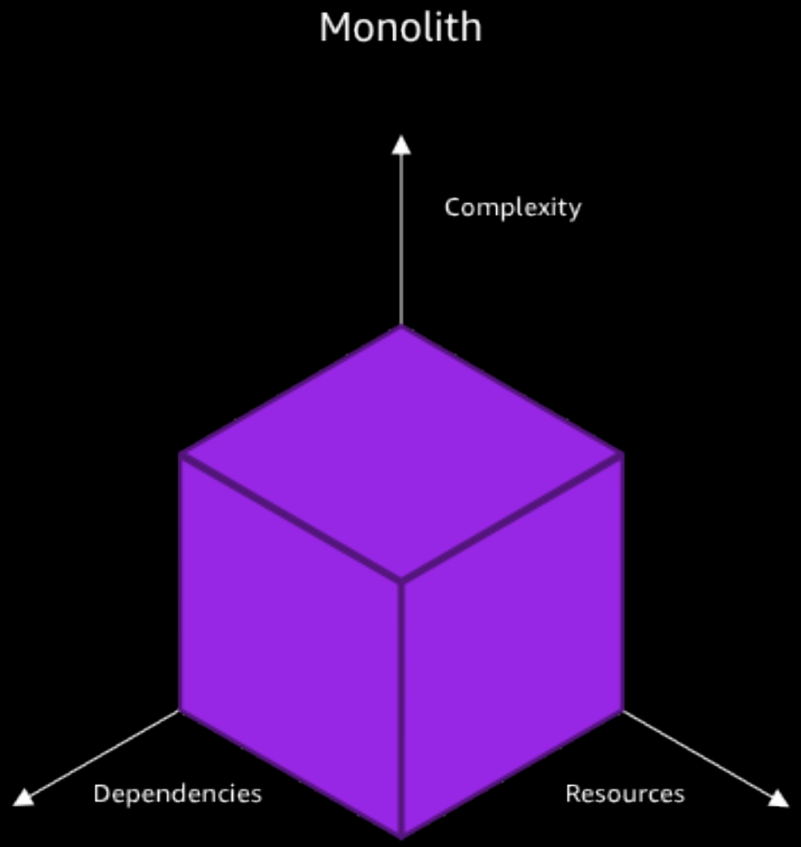
General guidance

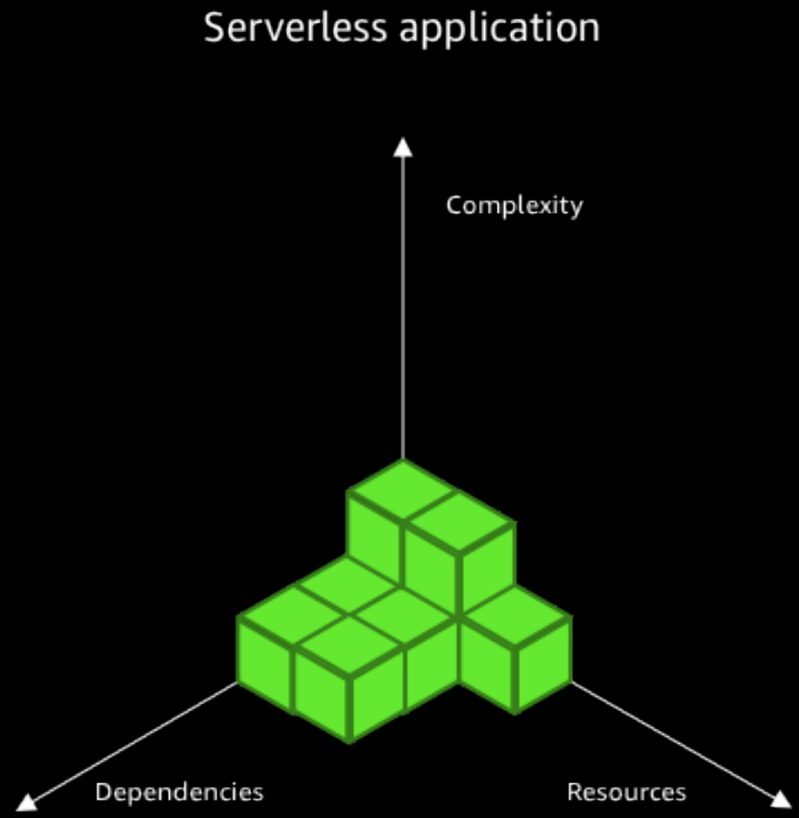
AWS re:Invent

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Finer-grained control reduces your risk

Consider the three dimensions of application risk





Don’t give away these benefits

- Don’t assign overly broad permissions
- Don’t build monolithic Lambda functions
- Don’t allow CRUD when “Read plus Write” will do
- Don’t access data you don’t need
- Don’t include unnecessary dependencies

Service-specific guidance

AWS re:Invent

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

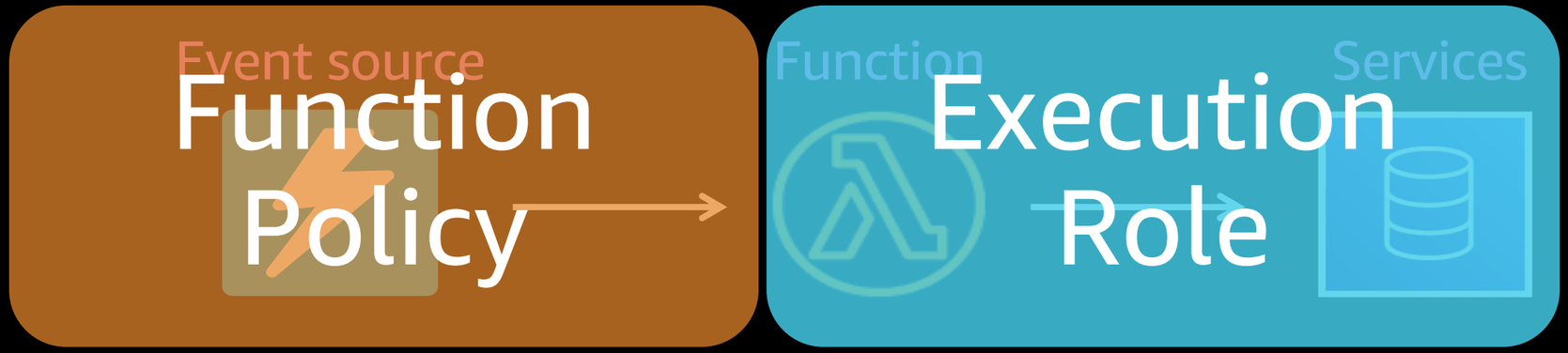
AWS Lambda

Function policies

- “Actions on bucket X can invoke Lambda function Z"
- Resource policies allow for cross-account access
- Used for sync and async invocations

Execution role

- “Lambda function A can read from DynamoDB table B”
- Define which AWS resources/API calls this function can access via IAM
- Used in streaming invocations



AWS Lambda – Function policy

Created implicitly by AWS SAM when you attach events

This AWS SAM template allows Amazon API Gateway to invoke the *saveToFreshTracksDatabaseTable* Lambda function

```
saveToFreshTracksDatabaseTable:
  Type: AWS::Serverless::Function
  Properties:
    Events:
      HttpPost:
        Type: Api
        Properties:
          Auth:
            Authorizer: MyLambdaTokenAuthorizer
          Path: '/activity'
          Method: post
          RestApiId: !Ref FreshTracksAPI
```

AWS Lambda – Execution role

Created explicitly by you when you define your function.

This AWS SAM template allows the Lambda function *saveToDynamoDB* to write to the Amazon DynamoDB table *UserDatabaseTable*.

```
saveToDynamoDB:
  Type: AWS::Serverless::Function
  Properties:
    Policies:
      - DynamoDBWritePolicy:
          TableName: !Ref UserDatabaseTable
```

AWS SAM policy templates

- Included in the AWS Serverless Application Model (AWS SAM)
- Help you quickly scope permissions to the resources used by your application
- Open source: submit pull requests and issues at

  s12d.com/sam-repo

AWS SAM policy templates

For more information and a complete list, visit s12d.com/sam-policy

```yaml
MyFunction:
  Type: AWS::Serverless::Function
  Properties:
    Policies:
      - SQSPollerPolicy: # Policy template with placeholder value
          QueueName:
            !GetAtt MyQueue.QueueName
      - CloudWatchPutMetricPolicy: {}  # Policy template with no placeholder value
      - DynamoDBWritePolicy:
          TableName:
            !Ref MyTable
```

AWS SAM policy templates

These two lines:

```
- DynamoDBReadPolicy:
    TableName: !Ref MyTable
```

Become this complete policy:

```
"Statement": [
  {
    "Effect": "Allow",
    "Action": [
      "dynamodb:GetItem",
      "dynamodb:Scan",
      "dynamodb:Query",
      "dynamodb:BatchGetItem",
      "dynamodb:DescribeTable"
    ],
    "Resource": [
      {
        "Fn::Sub": [
          "arn:${AWS::Partition}:dynamodb:${AWS::Region}:${AWS::AccountId}:table/${tableName}",
          {
            "tableName": {
              "Ref": "TableName"
            }
          }
        ]
      },
      {
        "Fn::Sub": [
          "arn:${AWS::Partition}:dynamodb:${AWS::Region}:${AWS::AccountId}:table/${tableName}/index/*",
          {
            "tableName": {
              "Ref": "TableName"
            }
          }
        ]
      }
    ]
  }
]
```

Amazon API Gateway

IAM permissions

- Use IAM policies and AWS credentials to grant access

Lambda authorizers

- Use a Lambda function for custom authorization logic

Amazon Cognito user pools

- Create a fully managed user management system

Resource policies

- Can restrict based on IP, VPC, AWS account ID



Amazon API Gateway

Amazon DynamoDB

Start with the AWS SAM policy templates

- *DynamoDBReadPolicy* for read-only
- *DynamoDBWritePolicy* for creates and updates
- *DynamoDBStreamReadPolicy* to attach to streams
- Avoid *DynamoDBCrudPolicy* whenever possible
- Command-query responsibility separation (CQRS)



Amazon DynamoDB

Allows for fine-grained access via the IAM condition *dynamodb:LeadingKeys*

Amazon S3

S3 buckets are not public by default.
In general, **don't change this!**



Amazon Simple Storage Service

Again, take advantage of AWS SAM policy templates:

- *S3ReadPolicy* for retrieving data
- *S3WritePolicy* for storing data
- Avoid using *S3CrudPolicy* and *S3FullAccessPolicy* whenever possible

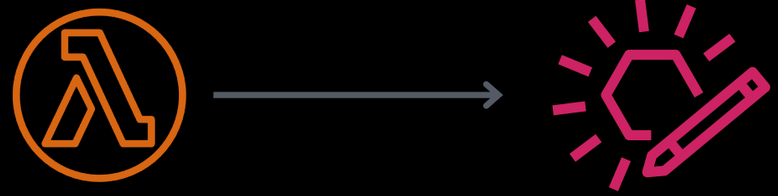
Use S3 Access Points for even greater access control over your buckets

Amazon EventBridge

IAM offers permissions for inbound and outbound operations

Inbound operations determine what principals can place events onto event buses and define rules and targets:

- *events:PutEvents*
- *events:PutRule*
- *events:PutTargets*



Lambda function

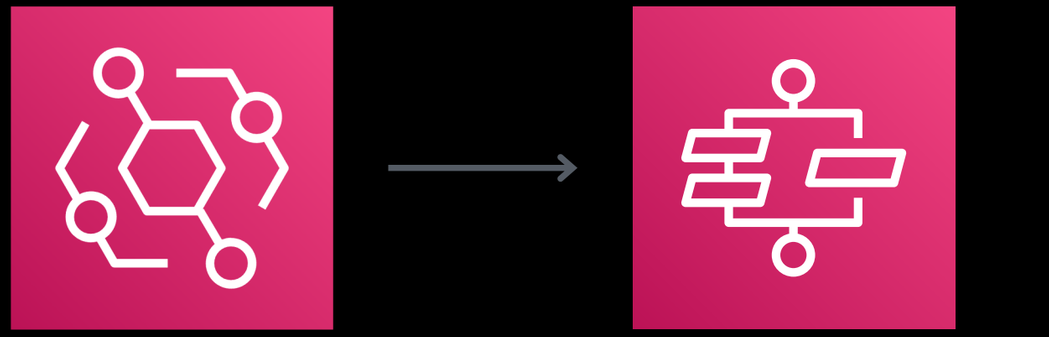
Custom event bus

```
PublishEventsFunction:
  Type: AWS::Serverless::Function
  Properties:
    CodeUri: publish-events/
    Handler: app.lambdaHandler
    Runtime: nodejs12.x
    Policies:
      - Statement:
          - Effect: Allow
            Action:
              - events:PutEvents
            Resource:
              - !GetAtt EventBus.Arn
```

Amazon EventBridge

IAM offers permissions for inbound and outbound operations

Outbound permissions are determined by the receiving resource



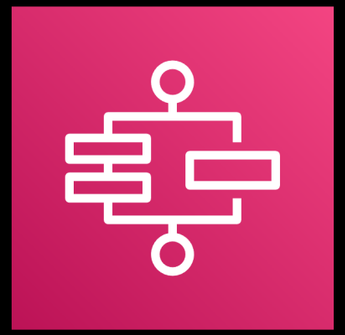

```yaml
InvokeWorkflowRole:
  Type: AWS::IAM::Role
  Properties:
    AssumeRolePolicyDocument:
      Version: "2012-10-17"
      Statement:
        - Effect: Allow
          Principal:
            Service:
              - events.amazonaws.com
          Action: sts:AssumeRole
    Policies:
      - PolicyName: InvokeCustomerWorkflowsPolicy
        PolicyDocument:
          Version: "2012-10-17"
          Statement:
            - Effect: Allow
              Action:
                - states:StartExecution
              Resource:
                - !Ref MyExpressWorkflow
```

AWS Step Functions

- Data in AWS Step Functions is encrypted at rest
- All data that passes between Step Functions and integrated services is encrypted using Transport Layer Security (TLS)

IAM governs Step Functions executions and invocations

- Special consideration for service integrations
  - Run a Job (*.sync*)
  - Wait for Callback (*.waitForTaskToken*)



Standard Workflows



Express Workflows

Considering trade-offs



AWS re:Invent

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Simplicity wins

- YAGNI (you aren’t gonna need it) applies – OFTEN
- Single-responsibility principle
- Complexity is one of our risk axes

Simplicity before complexity is worthless.

Simplicity beyond complexity is priceless.



AWS re:Invent

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Essential (forced) complexity

- Real business logic isn’t simple
- Governance and compliance
- High-risk threat models

Real-world simplicity

Where possible

- Only handle the ”happy path” inside your Lambda function
- Use Step Functions workflows for error handling
- Ruthlessly decompose your business logic
- Eliminate code with service integrations

Compliance

- Compliance-ready for SOC, PCI, FedRAMP, HIPAA, and others

Service	SOC	PCI	ISO	FedRAMP	HIPAA BAA
AWS Lambda	✅	✅	✅	✅	✅
Amazon API Gateway	✅	✅	✅	✅	✅
Amazon DynamoDB	✅	✅	✅	✅	✅
Amazon S3	✅	✅	✅	✅	✅
Amazon EventBridge	✅	✅	✅	✅	✅
AWS Step Functions	✅	✅	✅	✅	✅

Learn more at s12d.com/compliance

Summary

Serverless application security is

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

This is only a start! AWS provides a number of solutions to secure your applications. For more, visit

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



Thank you!

Rob Sutter

Sr. Developer Advocate
AWS
Twitch: /robsutter
Twitter: @rts_rob

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.



Please complete the session survey

AWS re:Invent

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.