Patterns for Efficient Serverless Development

Yan Cui

@theburningmonk

http://theburningmonk.com

AWS user since 2010

Yan Cui

@theburningmonk

http://theburningmonk.com

Developer Advocate @ lumigo

Yan Cui

@theburningmonk

http://theburningmonk.com

aws serverless HERO

Independent Consultant

Klarna. Intelliflo TotallyMoney Driver & Vehicle Licensing Agency TOYOTA YNAP

Zava JustGiving SB Simply Business DAZN lumigo XING

InDebted resolver SPARKOL tyke.io kits SOLVE grid

training

advise

delivery

Efficient Serverless Development requires

1.

2.

3.

Efficient Serverless Development requires

1. Testing

2. Deployment

3. Environments

Testing

“remocal” testing for Lambda functions

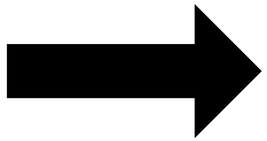
Local testing

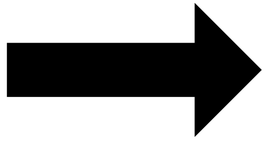
Runs code locally ✓

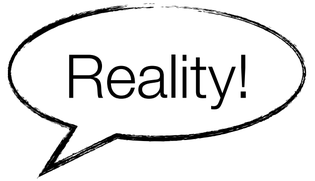
Can use debugger ✓

Change code without deployment ✓

FAST FEEDBACK!

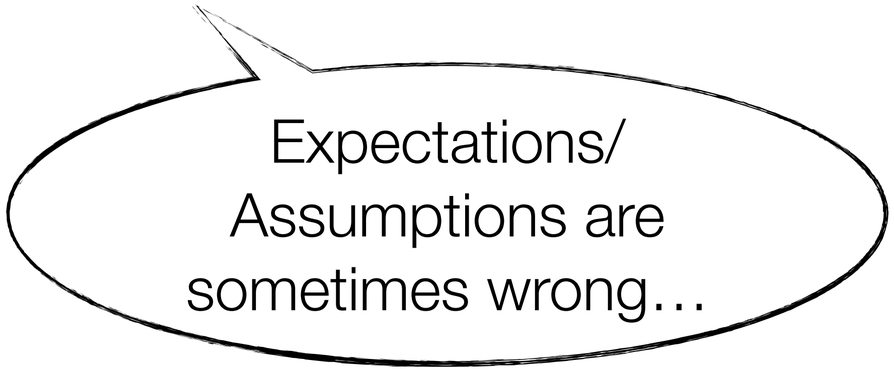
Test against real thing  “does it work?”

Test against mocks  “does it do what I expect?”



Test against real thing → **“does it work?”**

Test against mocks → **“does it do what I expect?”**



SNOWMEN

Local testing

Runs code locally

Can use debugger

Change code without deployment

FAST FEEDBACK!

Limited coverage

Prone to false positives

LOW CONFIDENCE

Your application consists of more than just your code

Your application consists of more than just your code
Your job is to ensure all of it works

Local testing

Runs code locally

Can use debugger

Change code without deployment

FAST FEEDBACK!

Limited coverage

Prone to false positives

LOW CONFIDENCE

Remote testing

Test in the cloud

Realistic tests

Better coverage

HIGH CONFIDENCE!

Local testing

Runs code locally ✓

Can use debugger ✓

Change code without deployment

FAST FEEDBACK!

Limited coverage ✗

Prone to false positives ✗

LOW CONFIDENCE

Remote testing

Test in the cloud ✓

Realistic tests ✓

Better coverage ✓

HIGH CONFIDENCE!

Slow deployments ✗

Every change needs deploying… ✗

SLOW FEEDBACK…

Local testing

Runs code locally ✓

Can use debugger ✓

Change code without deployment

FAST FEEDBACK!

Limited coverage ✗

Prone to false positives ✗

LOW CONFIDENCE

Remote testing

Test in the cloud ✓

Realistic tests ✓

Better coverage ✓

HIGH CONFIDENCE!

Slow deployments ✗

Every change needs deploying… ✗

SLOW FEEDBACK…


Local testing
Remote testing

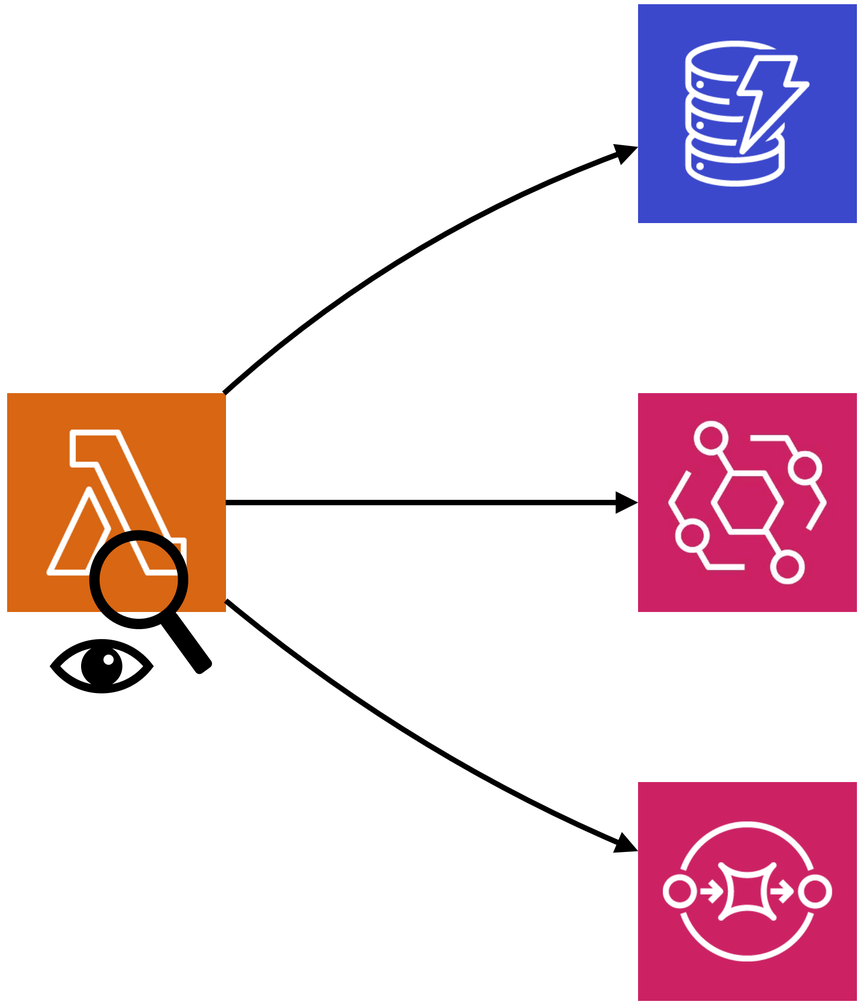
REMOCAL testing

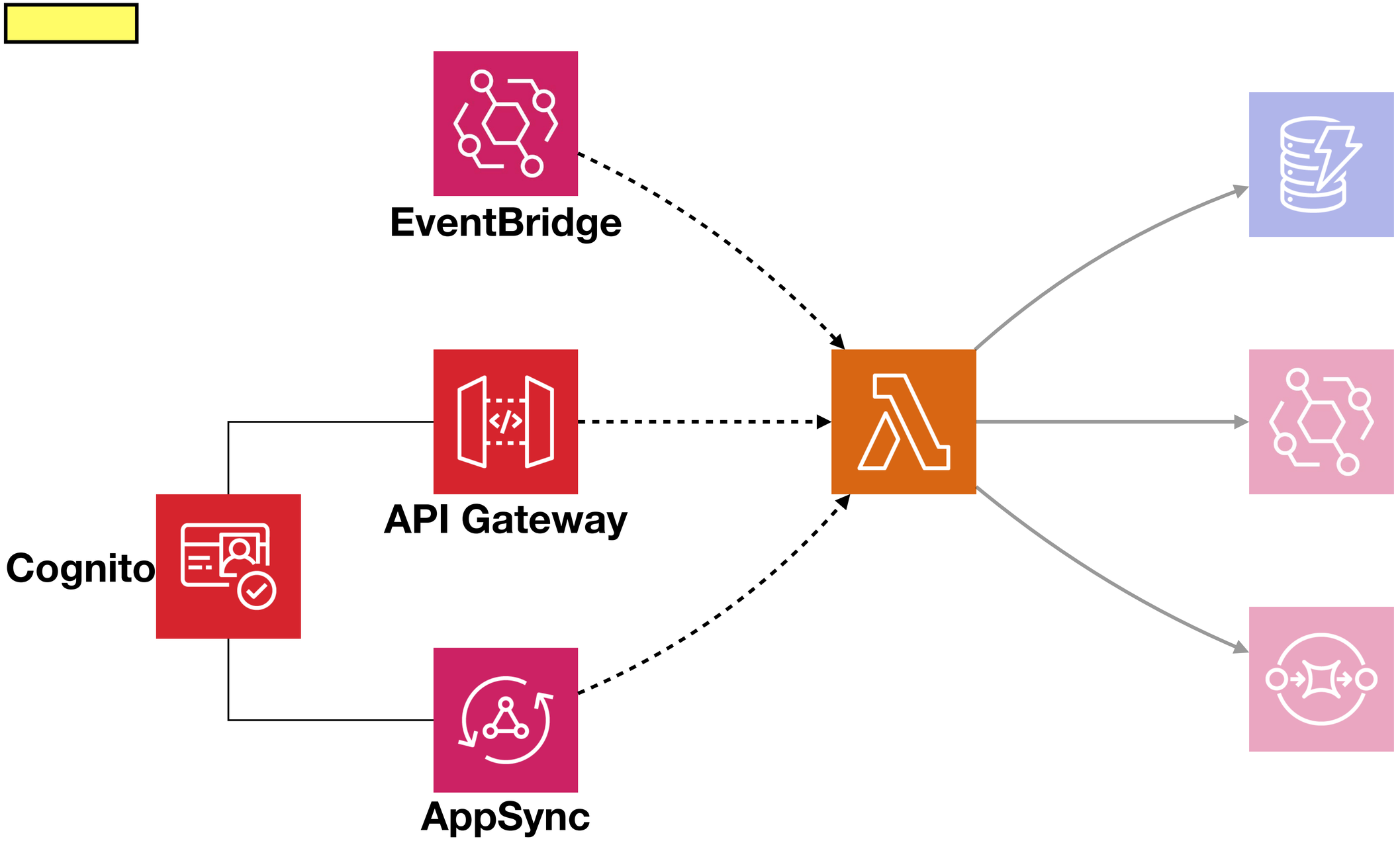
Runs code locally, talk to **real** AWS services ✓

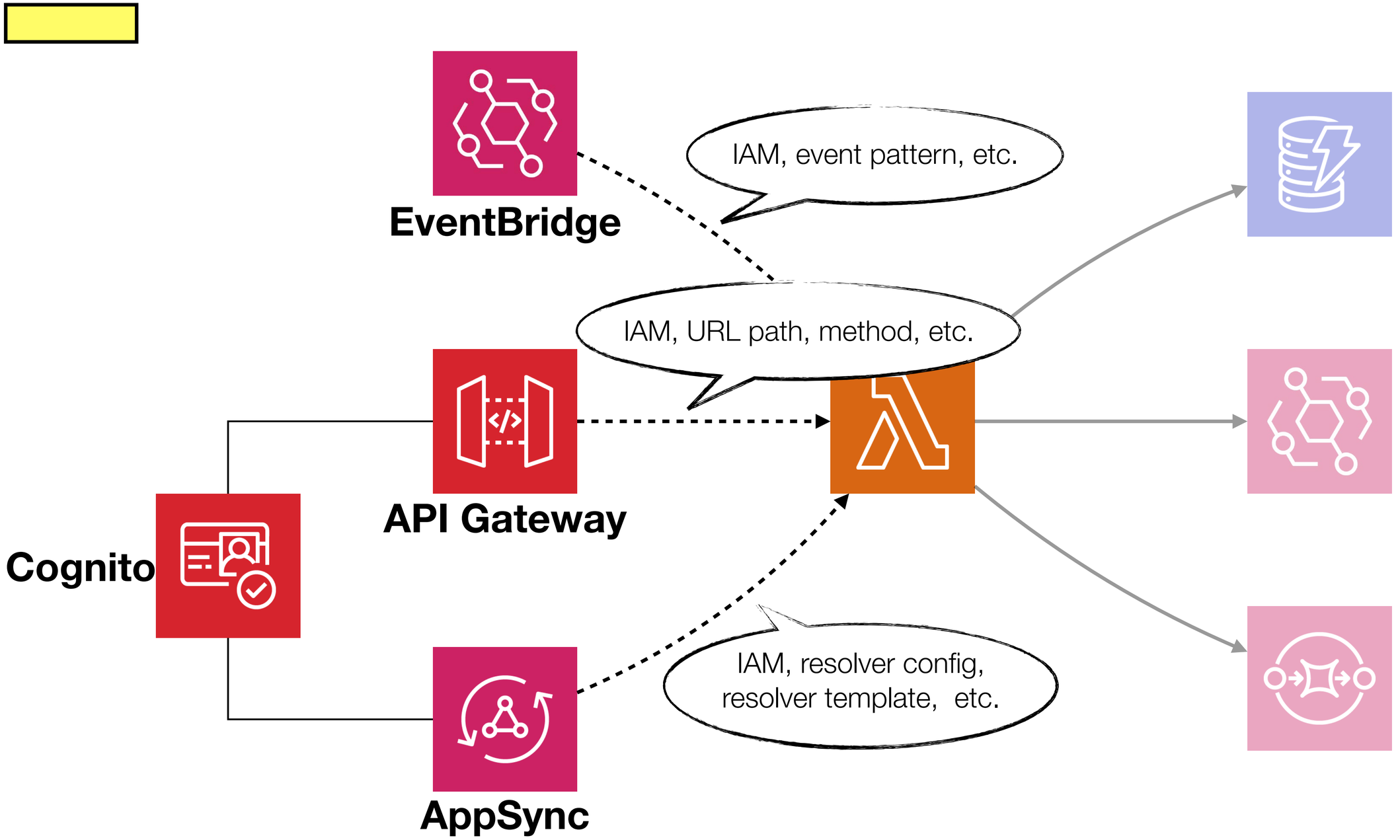
Can use debugger ✓

Change code without deployment ✓

Realistic tests ✓

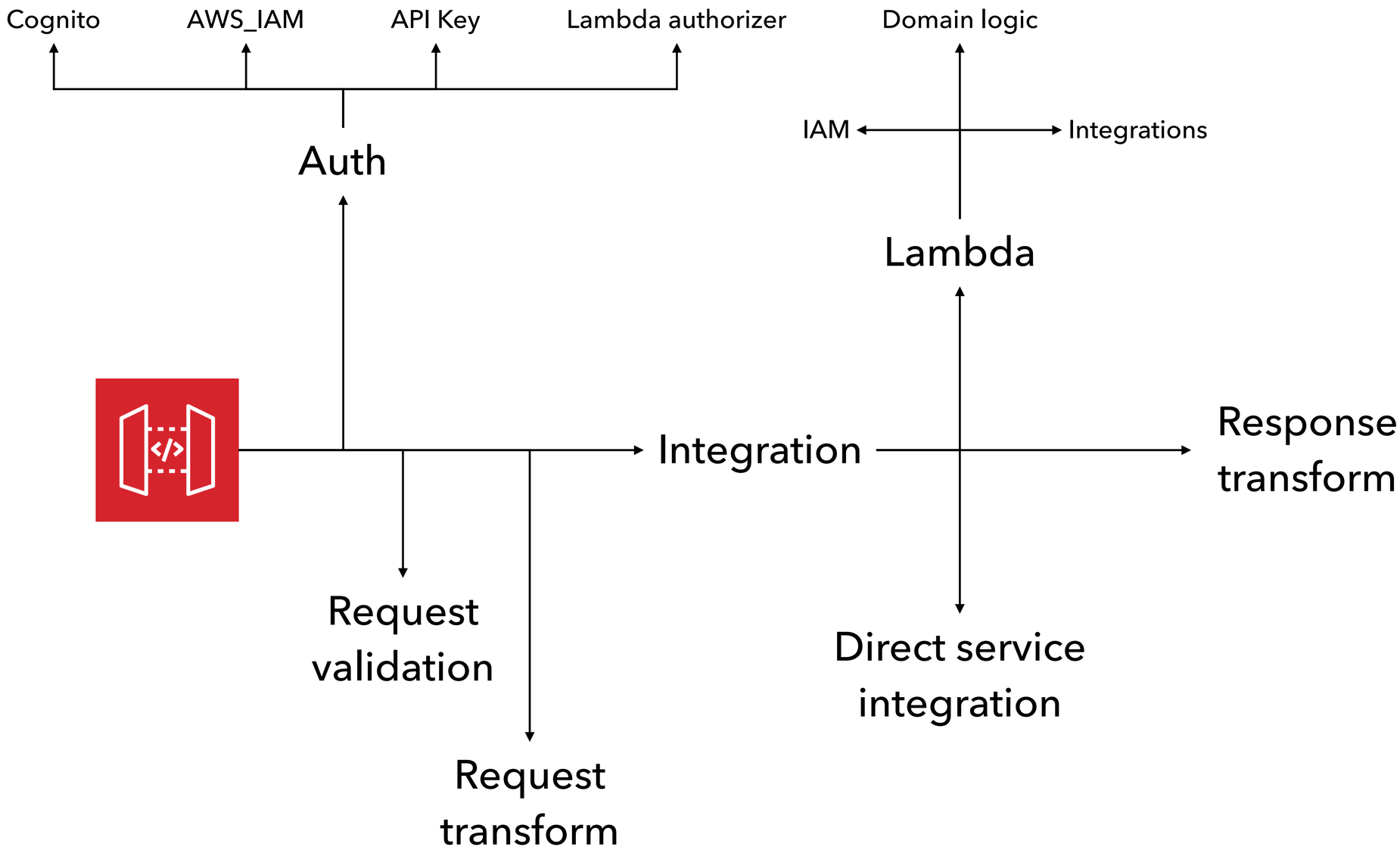


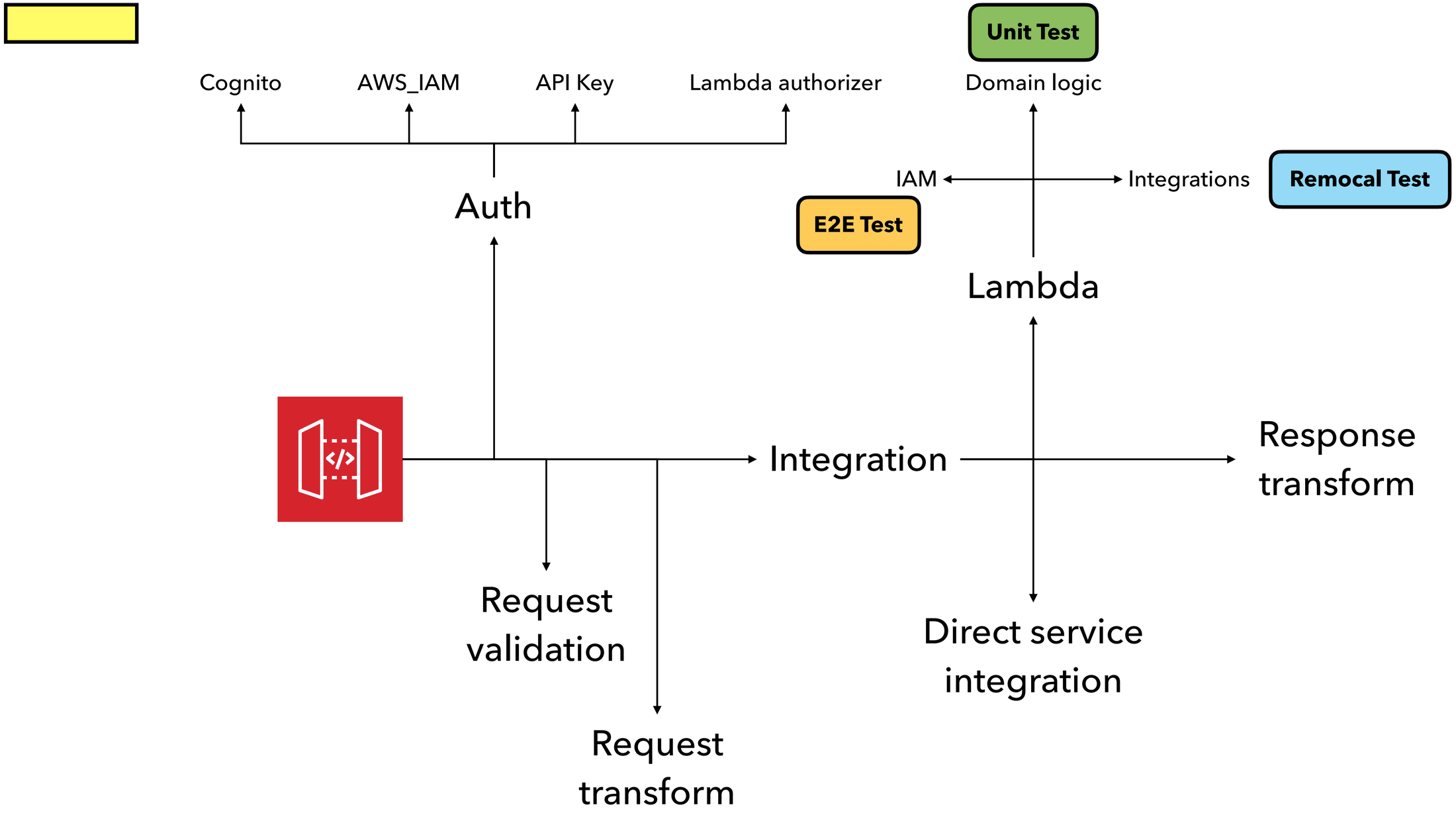

EventBridge
API Gateway
Cognito
AppSync

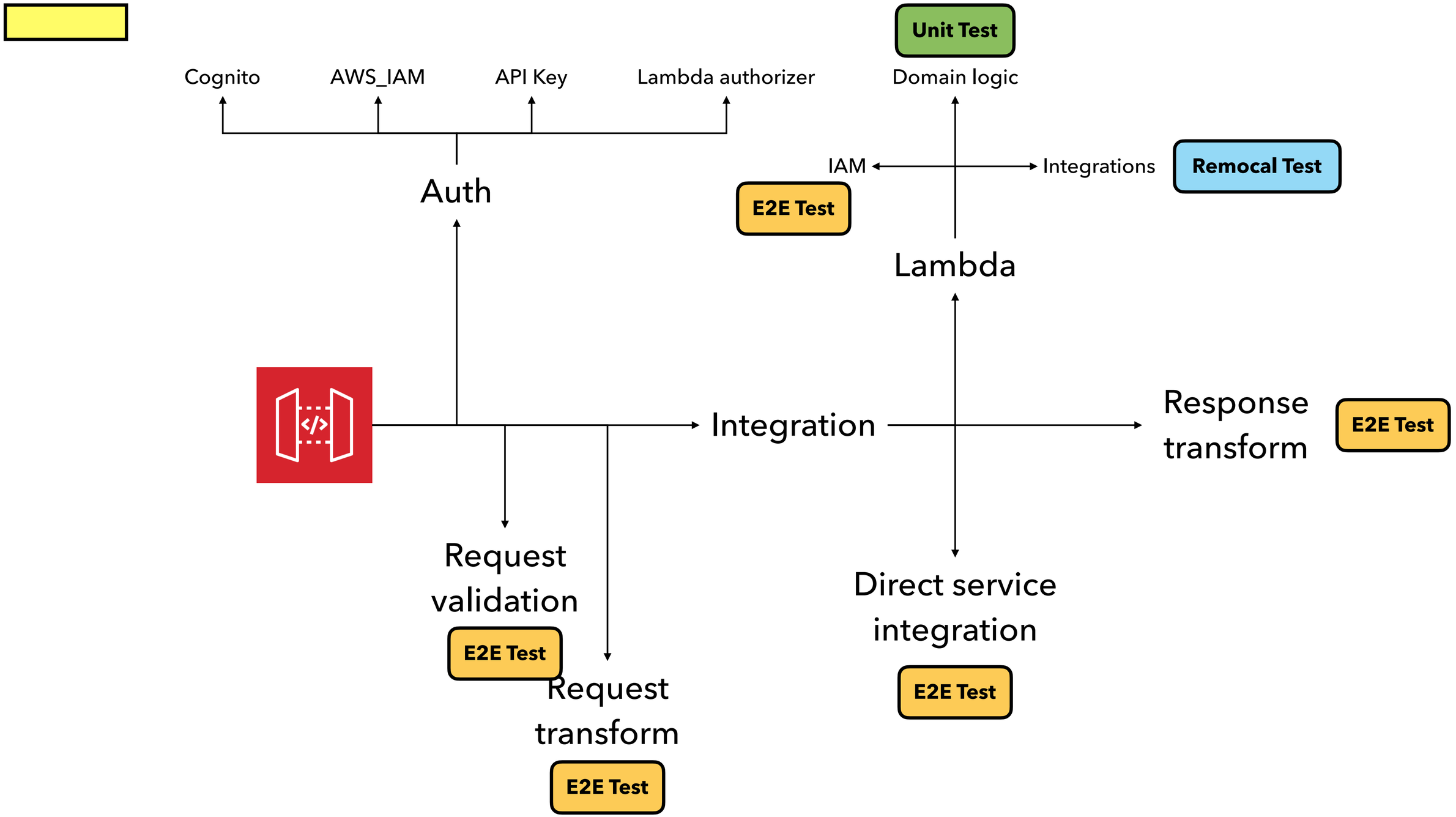

EventBridge
IAM, event pattern, etc.
IAM, URL path, method, etc.
API Gateway
Cognito
IAM, resolver config,
resolver template, etc.
AppSync

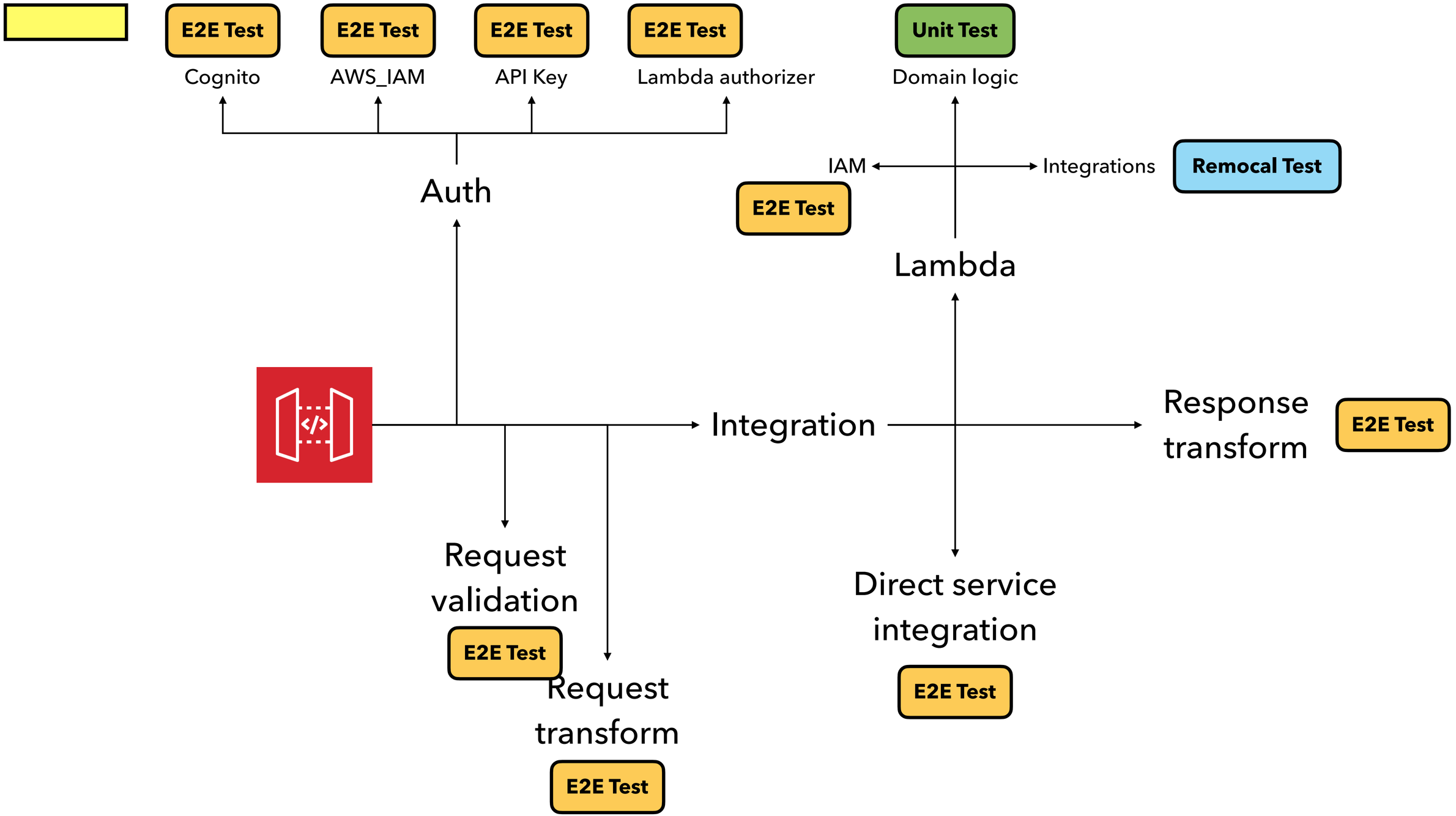
Your application consists of more than just your code

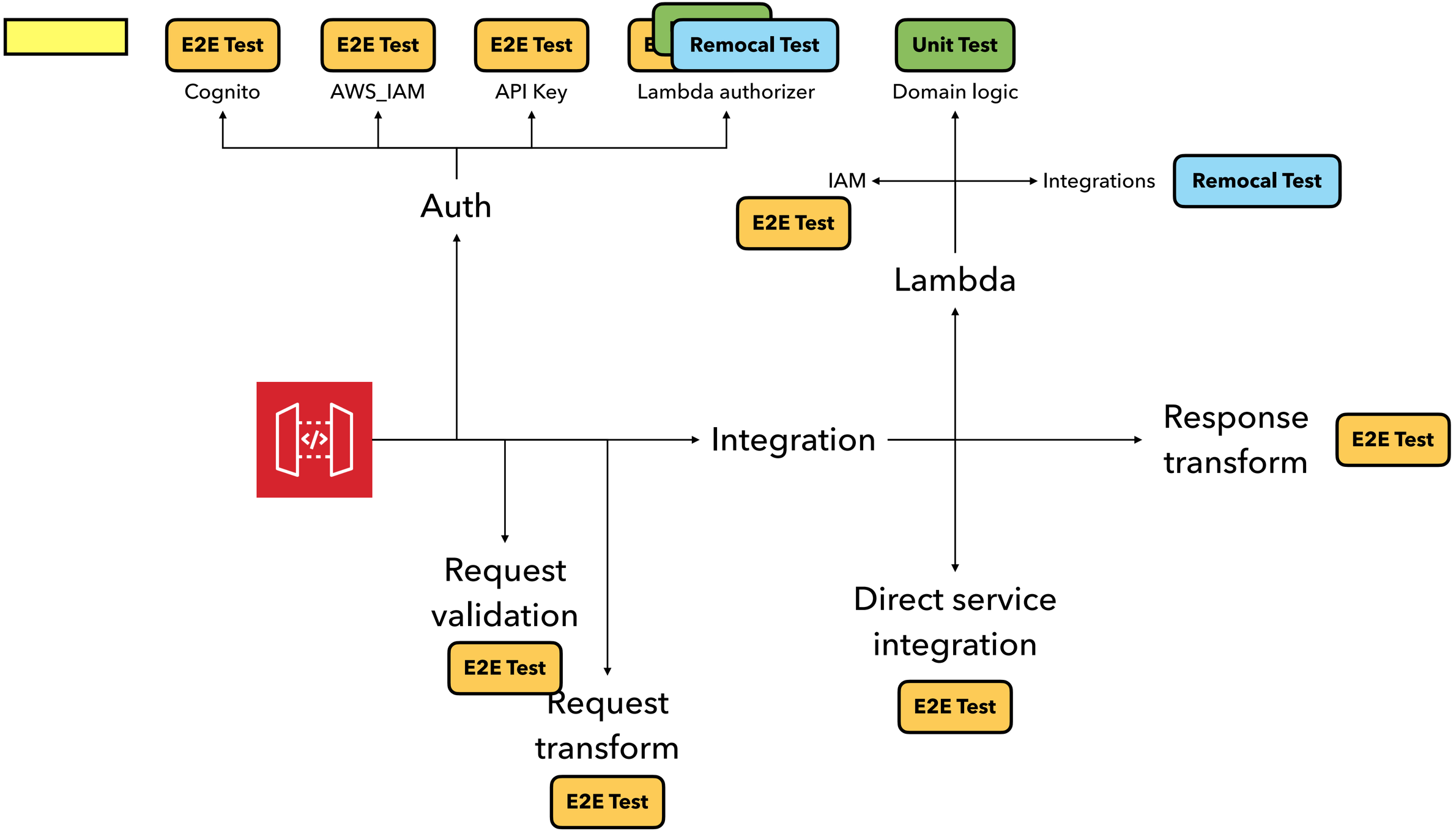
Your job is to ensure all of it works


Cognito
AWS_IAM
API Key
Lambda authorizer
Auth
Domain logic
IAM
Integrations
Lambda
Integration
Response transform
Request validation
Request transform
Direct service integration


Unit Test
Cognito
AWS_IAM
API Key
Lambda authorizer
Domain logic
IAM
Integrations
Remocal Test
E2E Test
Auth
Lambda
Integration
Response transform
Request validation
Direct service integration
Request transform


Unit Test
Cognito
AWS_IAM
API Key
Lambda authorizer
Domain logic
IAM
Integrations
Remocal Test
E2E Test
Auth
Lambda
Integration
Response transform
E2E Test
Request validation
E2E Test
Request transform
E2E Test
Direct service integration
E2E Test


E2E Test
E2E Test
E2E Test
E2E Test
Unit Test
Cognito
AWS_IAM
API Key
Lambda authorizer
Domain logic
IAM
Integrations
Remocal Test
Auth
E2E Test
Lambda
Integration
Response transform
E2E Test
Request validation
E2E Test
Request transform
E2E Test
Direct service integration
E2E Test


E2E Test
E2E Test
E2E Test
Remocal Test
Unit Test
Cognito
AWS_IAM
API Key
Lambda authorizer
Domain logic
Auth
IAM
Integrations
Remocal Test
E2E Test
Lambda
Integration
Response transform
E2E Test
Request validation
E2E Test
Request transform
E2E Test
Direct service integration
E2E Test


DEMO TIME

REMOCAL testing

Runs code locally, talk to **real** AWS services ✓

Can use debugger ✓

Change code without deployment ✓

Realistic tests ✓

AWS resources need to be provisioned ✗

Ephemeral environments

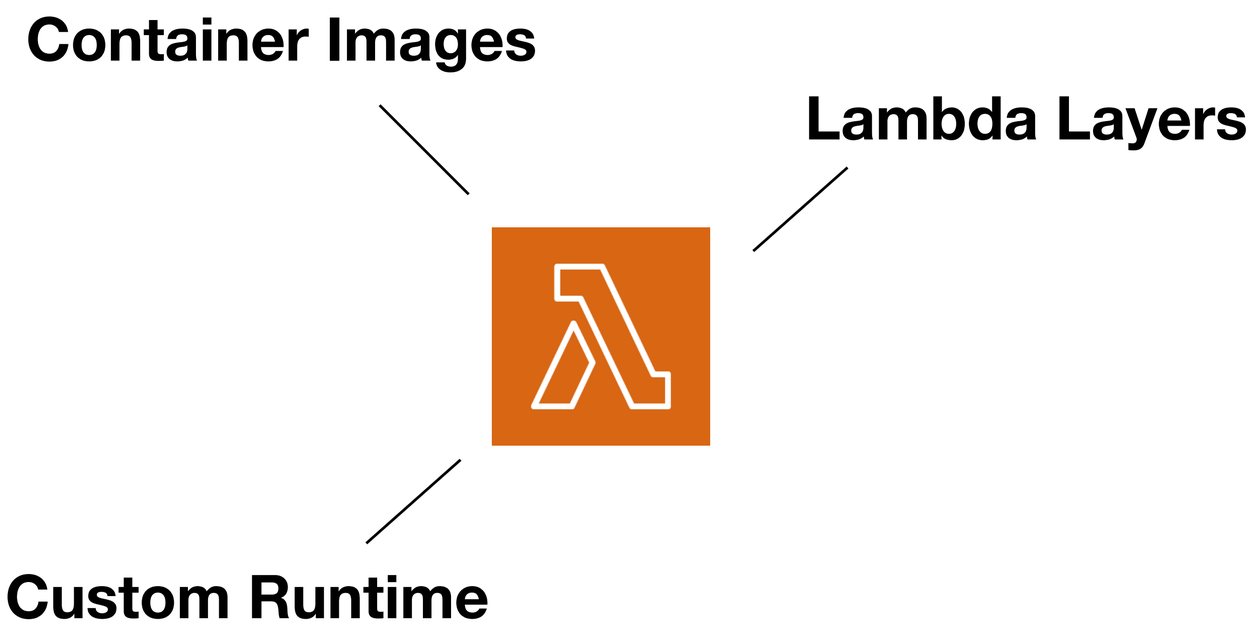
Deployment

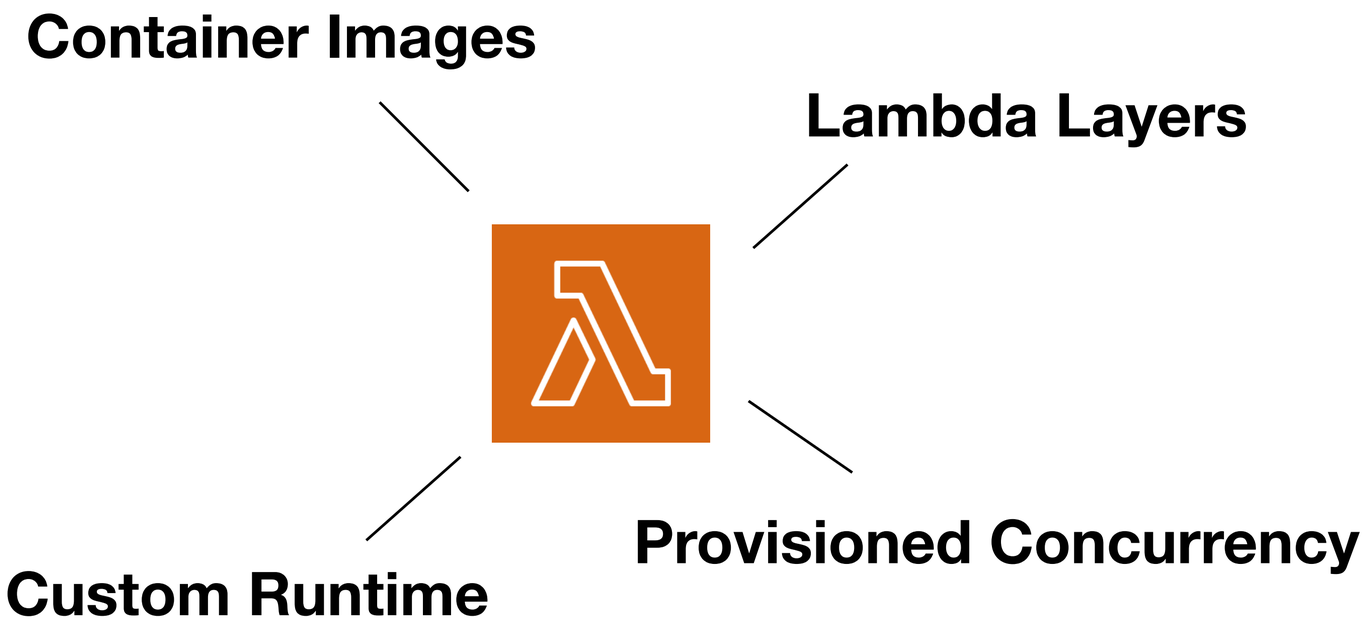
Keep deployments as simple as possible

Lambda Layers

Container Images

Lambda Layers


Container Images
Lambda Layers
Custom Runtime


Container Images
Lambda Layers
Custom Runtime
Provisioned Concurrency

You don’t have to use them!

DO NOT use Lambda Layers to share code

❌ Layers have no semantic versioning.

❌ Layers have no semantic versioning.

❌ Security scanning tools don't know about them and can't scan them.

❌ Layers have no semantic versioning.

❌ Security scanning tools don't know about them and can't scan them.

❌ You're limited to 5 layers per function.

❌ Layers have no semantic versioning.

❌ Security scanning tools don't know about them and can't scan them.

❌ You're limited to 5 layers per function.

❌ They still count towards Lambda's 250mb (unzipped) size limit.

❌ Layers have no semantic versioning.

❌ Security scanning tools don't know about them and can't scan them.

❌ You're limited to 5 layers per function.

❌ They still count towards Lambda's 250mb (unzipped) size limit.

❌ They make it harder to test your code locally with remocal testing.

❌ Layers have no semantic versioning.

❌ Security scanning tools don't know about them and can't scan them.

❌ You're limited to 5 layers per function.

❌ They still count towards Lambda's 250mb (unzipped) size limit.

❌ They make it harder to test your code locally with remocal testing.

❌ They don't really work with static languages.

❌ Layers have no semantic versioning.

❌ Security scanning tools don't know about them and can't scan them.

❌ You're limited to 5 layers per function.

❌ They still count towards Lambda's 250mb (unzipped) size limit.

❌ They make it harder to test your code locally with remocal testing.

❌ They don't really work with static languages.

❌ It takes more work to publish and update a package than NPM.

❌ Layers have no semantic versioning.

❌ Security scanning tools don't know about them and can't scan them.

❌ You're limited to 5 layers per function.

❌ They still count towards Lambda's 250mb (unzipped) size limit.

❌ They make it harder to test your code locally with remocal testing.

❌ They don't really work with static languages.

❌ It takes more work to publish and update a package than NPM.

❌ No tree-shaking and bundling

❌ Layers have no semantic versioning.

❌ Security scanning tools don't know about them and can't scan them.

❌ You're limited to 5 layers per function.

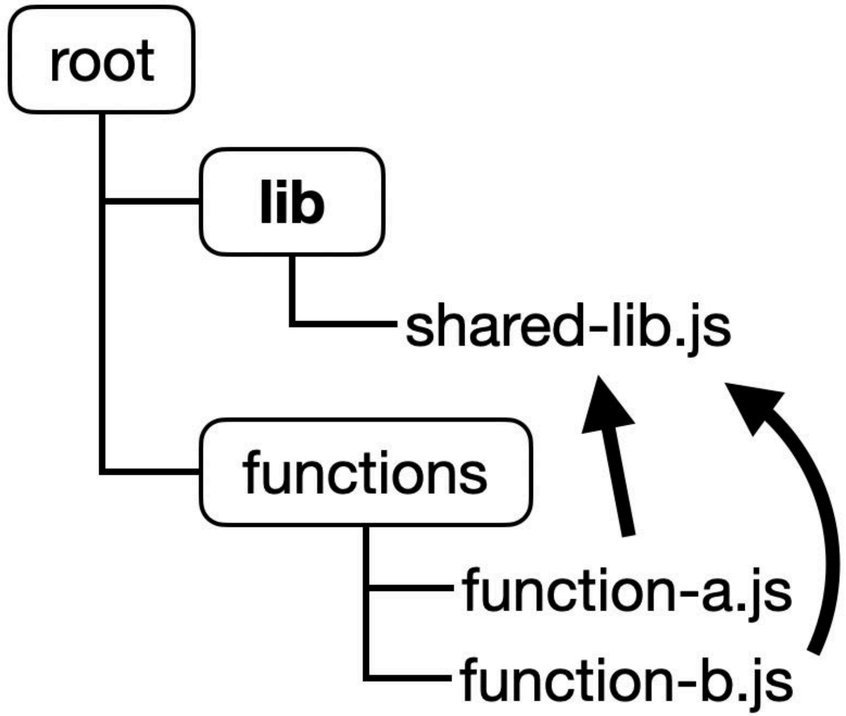
❌ They still count towards Lambda's 250mb (unzipped) size limit.

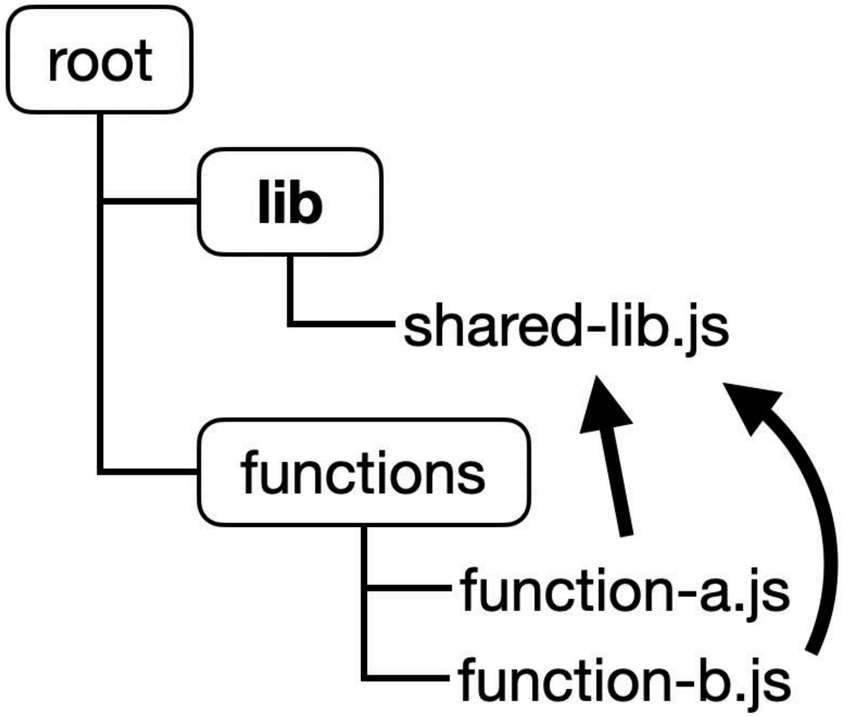
❌ They make it harder to test your code locally with remocal testing.

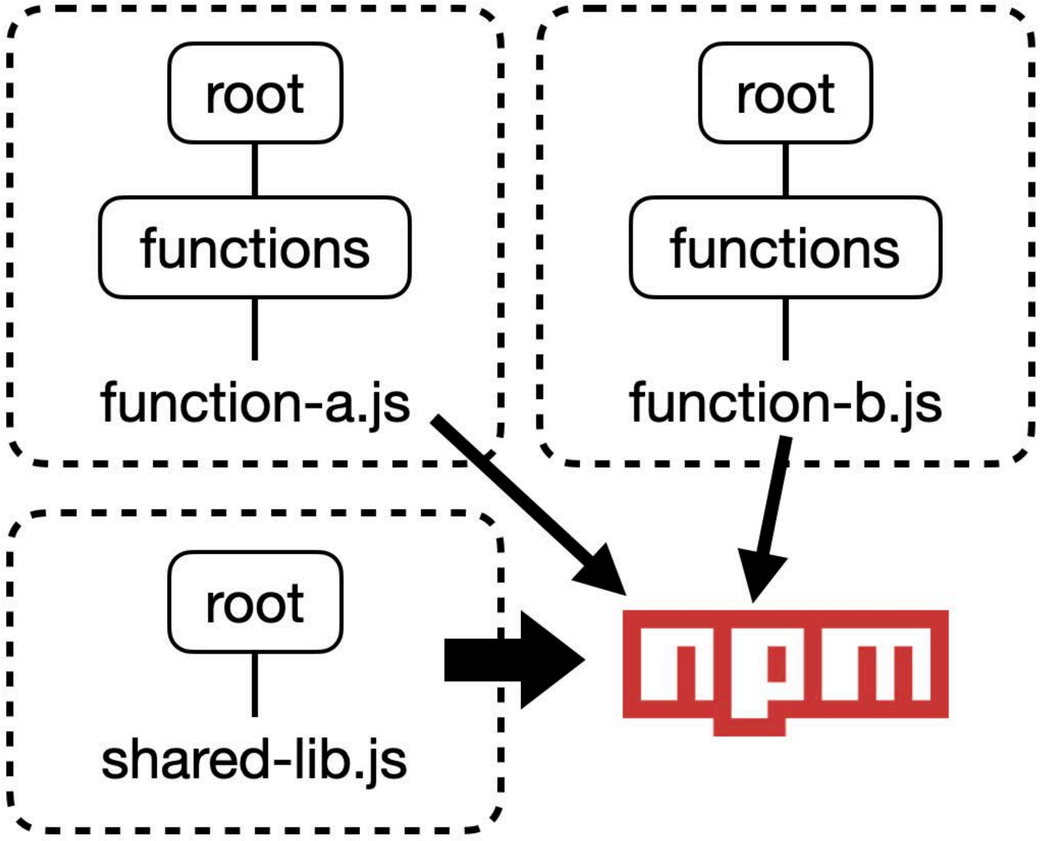
❌ They don't really work with static languages.

❌ It takes more work to publish and update a package than NPM.

❌ No tree-shaking and bundling

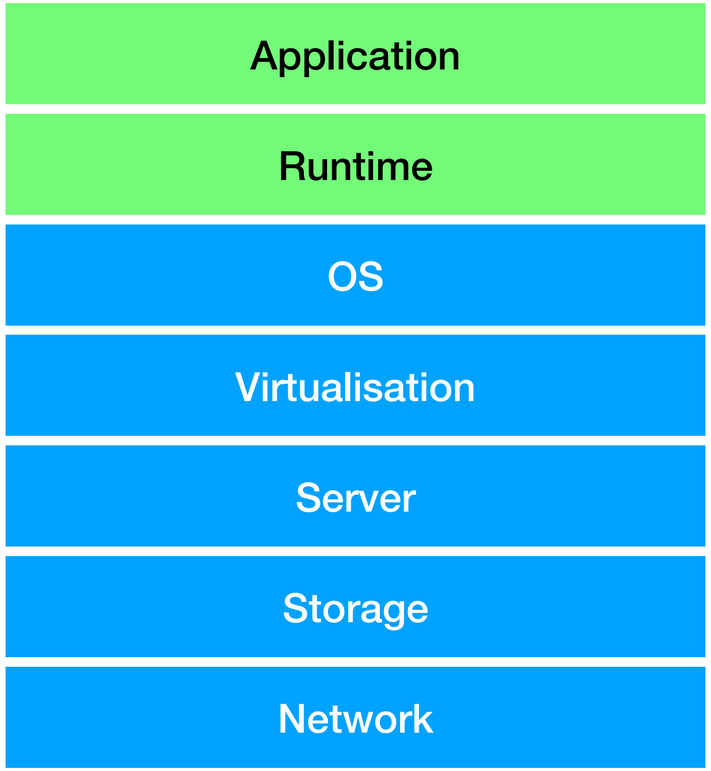

root
lib
shared-lib.js
functions
function-a.js
function-b.js


root
lib
shared-lib.js
functions
function-a.js
function-b.js

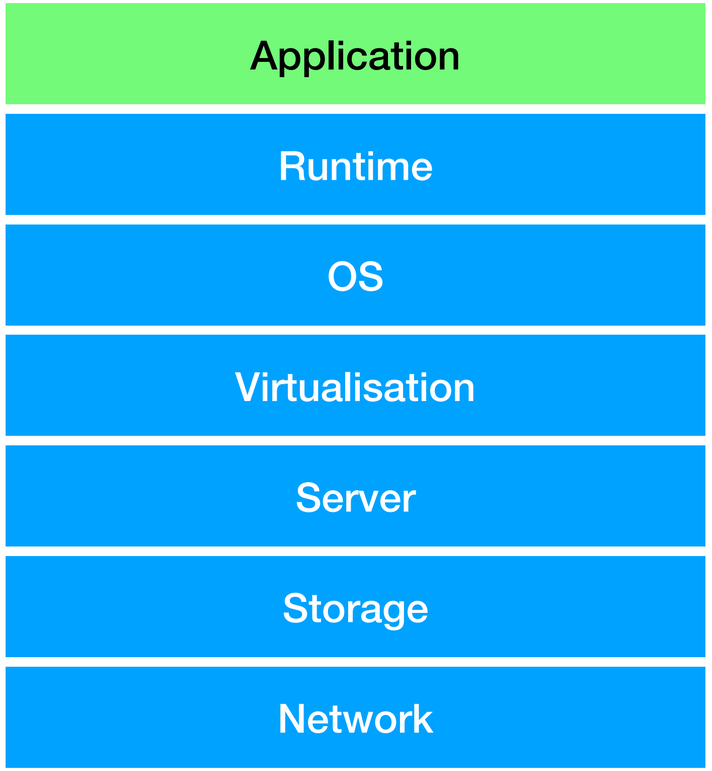

root
functions
function-a.js
root
functions
function-b.js
root
shared-lib.js
npm

Prefer zip files and managed runtimes

Container Image



Managed Runtime



You manage

Platform manages

- ✓ Don’t use Lambda Layers to share code.
- ✓ Use zip files and managed runtimes.

Environments

One account per stage, minimum

Accounts

dev

test

staging

prod

One account per team per stage for large organisations

Business critical workloads have separate accounts

Ephemeral environments

Step 1: npx sls deploy -s dev-my-feature

(creates a new “dev-my-feature” environment)

Step 1: npx sls deploy -s dev-my-feature

(creates a new “dev-my-feature” environment)

Step 2: Make code changes, iterate, run **remocal tests** against the “dev-my-feature” environment

Step 1: npx sls deploy -s dev-my-feature

(creates a new “dev-my-feature” environment)

Step 2: Make code changes, iterate, run **remocal tests** against the “dev-my-feature” environment

Step 3: Commit code and send PR

(CI pipeline runs all tests, etc.)

Step 1: npx sls deploy -s dev-my-feature

(creates a new “dev-my-feature” environment)

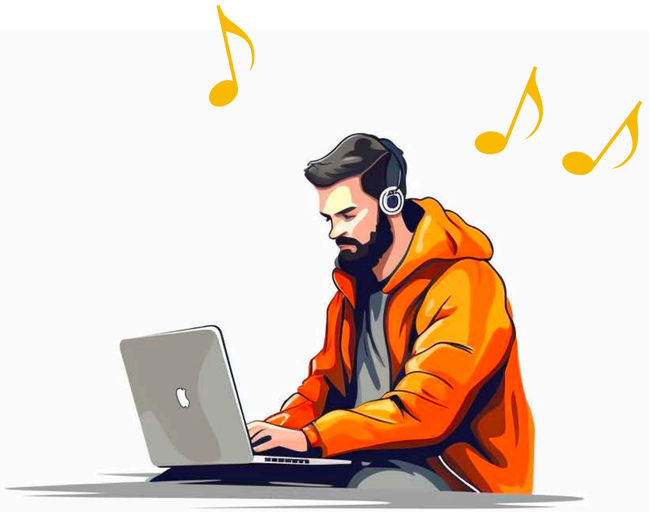
Step 2: Make code changes, iterate, run **remocal tests** against the “dev-my-feature” environment

Step 3: Commit code and send PR

(CI pipeline runs all tests, etc.)

Step 4: npx sls remove -s dev-my-feature

(destroys the ephemeral environment)



Step 1: npx sls deploy -s dev-my-feature

(creates a new “dev-my-feature” environment)

Step 2: Make code changes, iterate, run **remocal tests** against the “dev-my-feature” environment

Step 3: Commit code and send PR

(CI pipeline runs all tests, etc.)

Step 4: npx sls remove -s dev-my-feature

(destroys the ephemeral environment)

Insulated environment for development and testing

Insulated environment for development and testing

(avoids polluting shared dev/test/staging environments with test data)

No cost overhead with usage-based pricing

How to handle serverful resources when using ephemeral environments

AWS, Serverless / February 13, 2023

I'm a big fan of using ephemeral (or temporary) environments when I'm building serverless architectures. I have written about this practice before and I believe it's one of the most important practices that have co-evolved with the rise of serverless technologies.

It takes advantage of the pay-per-use pricing model offered by many serverless technologies such as Lambda and DynamoDB. You can create as many ephemeral environments as you need (resource limits permitting, of course). There are no extra charges for having these environments.

You can create an ephemeral environment when you start working on a feature and delete it when you're done. You can even create a fresh environment for every CI/CD run so you can test your code without worrying about polluting your dev/test environments with dummy test data.

To make it easy to create ephemeral environments for your services, I also prefer to keep stateful (e.g. databases) and stateless resources together. I wrote about this recently and addressed the most common counterarguments.

Using these two practices together has supercharged my development flow and I have seen these practices in organizations of all sizes.

However.

What about serverful resources?

Few things in life are black and white, and few practices are universally "best" for everyone.

https://theburningmonk.com/2023/02/how-to-handle-serverful-resources-when-using-ephemeral-environments

Step 1: npx sls deploy -s dev-my-featu
(creates a new “dev-my-feature” environ

Step 2: Make code changes, iterate, ru
remocal tests against the “dev-my-fea
environment

Step 3: Commit code and send PR
(CI pipeline runs all tests, etc.)

Step 4: npx sls remove -s dev-my-
(destroys the ephemeral environme

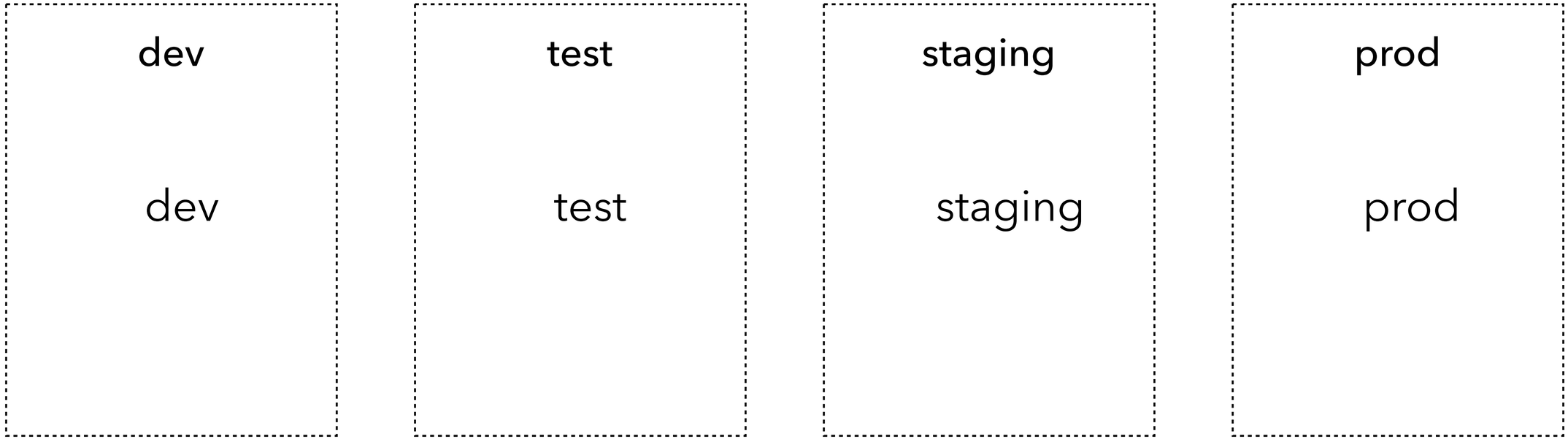
Step 1: npx sls deploy -s ci-<SHA>

Step 2: npm run tests:all

Step 3: npx sls remove -s ci-<SHA>

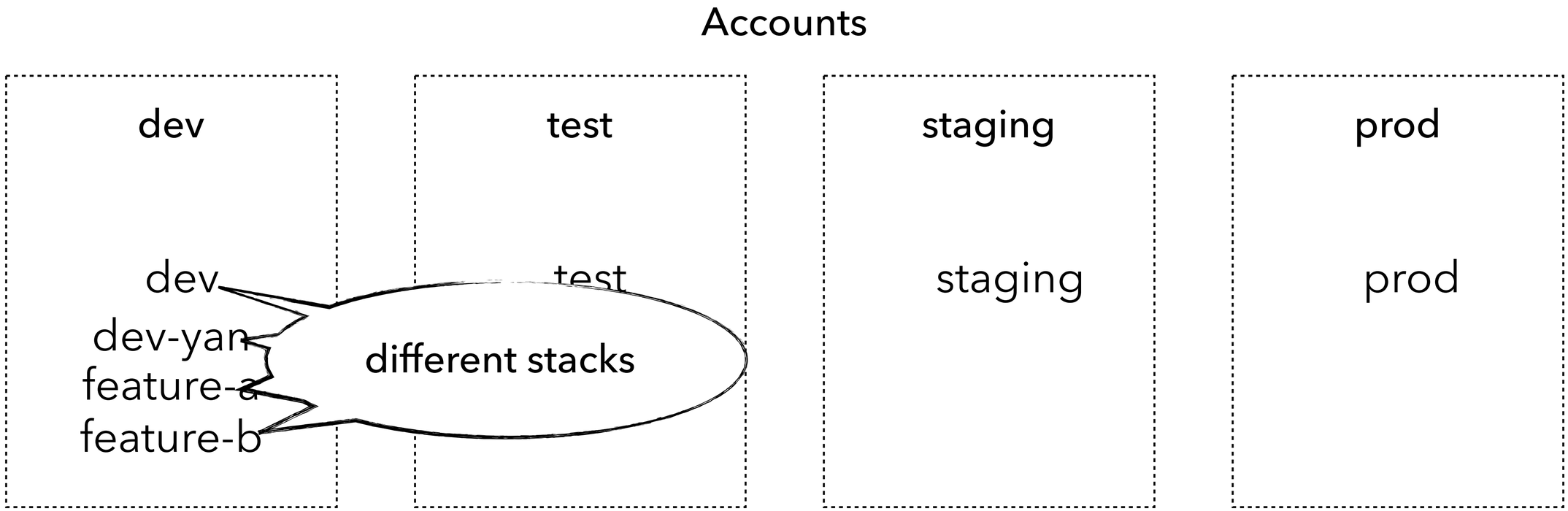
environment !== AWS account

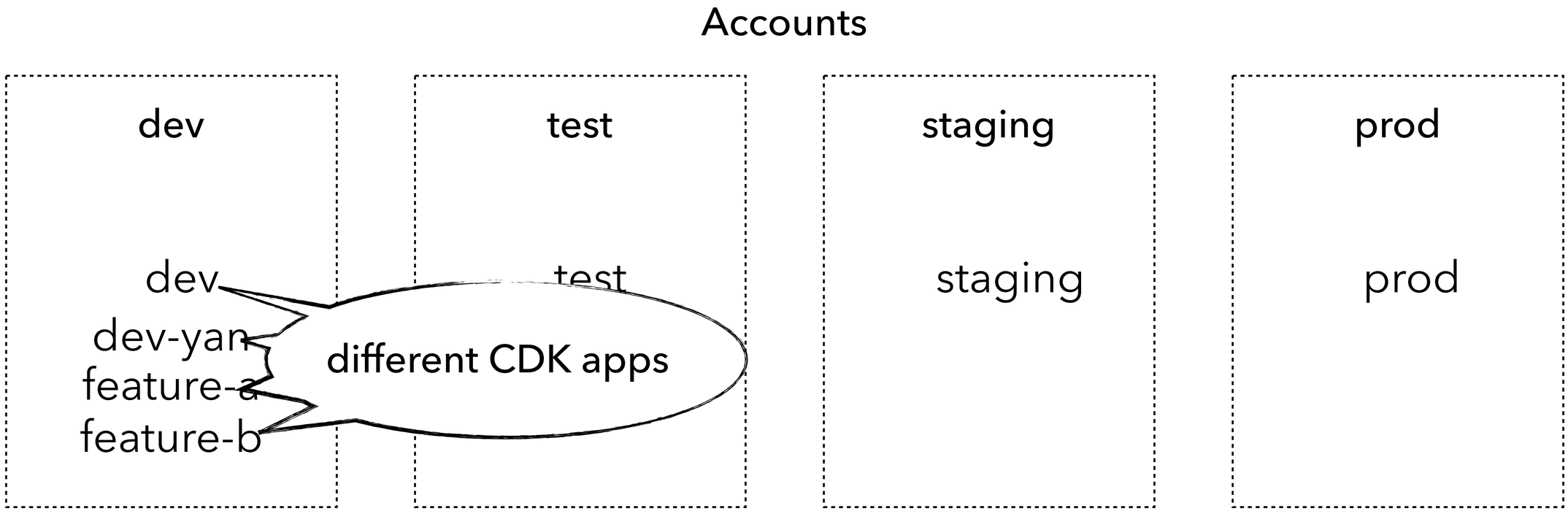
Accounts

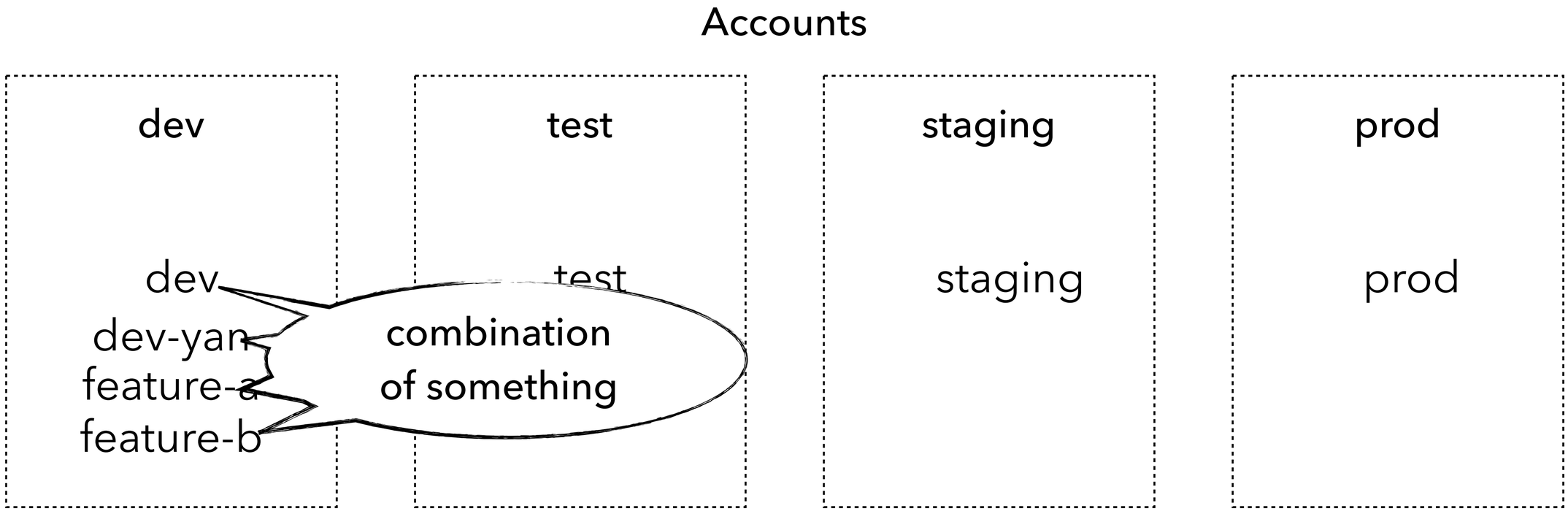


Accounts

dev	test	staging	prod
dev dev-yan feature-a feature-b	test	staging	prod


Accounts
dev
test
staging
prod
dev
test
staging
prod
dev-yan
feature-a
feature-b
different stacks


Accounts
dev
test
staging
prod
dev
dev-yan
feature-a
feature-b
test
staging
prod
different CDK apps


Accounts
dev
test
staging
prod
dev
dev-yan
feature-a
feature-b
test
combination
of something
staging
prod

“How do I make sure resource names don’t clash?”

“How do I make sure resource names don’t clash?”

#1: Don’t explicit name resources (unless you have to)

“How do I make sure resource names don’t clash?”

#1: Don’t explicit name resources (unless you have to)

#2: Include environment name in resource names

Works well with remocal testing

Efficient Serverless Development requires

1. Testing

2. Deployment

3. Environments

Questions?