



Continuous verification for serverless applications

Serverless WEEK

Gunnar Grosch
@gunnargrosch
October 30, 2020

“Testing to ensure that you can meet your availability goals is the only way you can have confidence that you will meet those goals”

Reliability Pillar
AWS Well-Architected Framework

Verification of applications

Verification of applications



Verification of applications



Conditions for verification

Errors in your code

Security policy errors

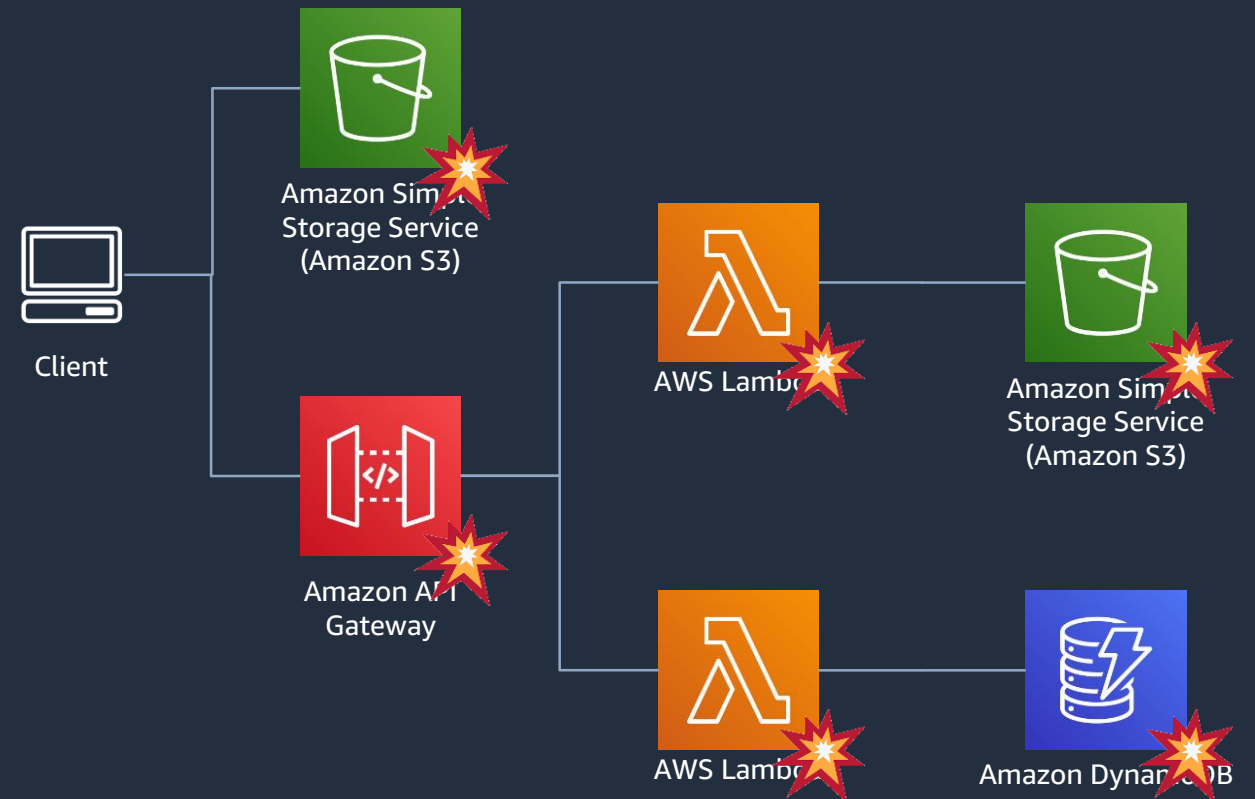
Service configuration errors

Function disk space failure

Downstream service issues

Concurrency and throttling

Latency



Tools for adding conditions

Chaos-lambda

Python

Failure-lambda

NodeJS

Failure-lambda NodeJS

NPM package for NodeJS Lambdas

Configuration using Parameter Store or AWS AppConfig

Several failure modes

- Latency
- Status code
- Exception
- Disk space
- Denylist

```
const failureLambda = require('failure-lambda')
exports.handler = failureLambda(async (event, context) => {
  ...
})

{
  "isEnabled": false,
  "failureMode": "latency",
  "rate": 1,
  "minLatency": 100,
  "maxLatency": 400,
  "exceptionMsg": "Exception message!",
  "statusCode": 404,
  "diskSpace": 100,
  "denylist": [
    "s3.*.amazonaws.com",
    "dynamodb.*.amazonaws.com"
  ]
}
```


Verification of applications



Verification example

Objective: My purchase API should respond in less than 400 ms

Measure: 400 ms threshold

Condition: 100-200 ms latency injection to function

Verify: Pass or fail



Verification example

Objective: I want 10 clicks per second

Measure: 10 clicks threshold

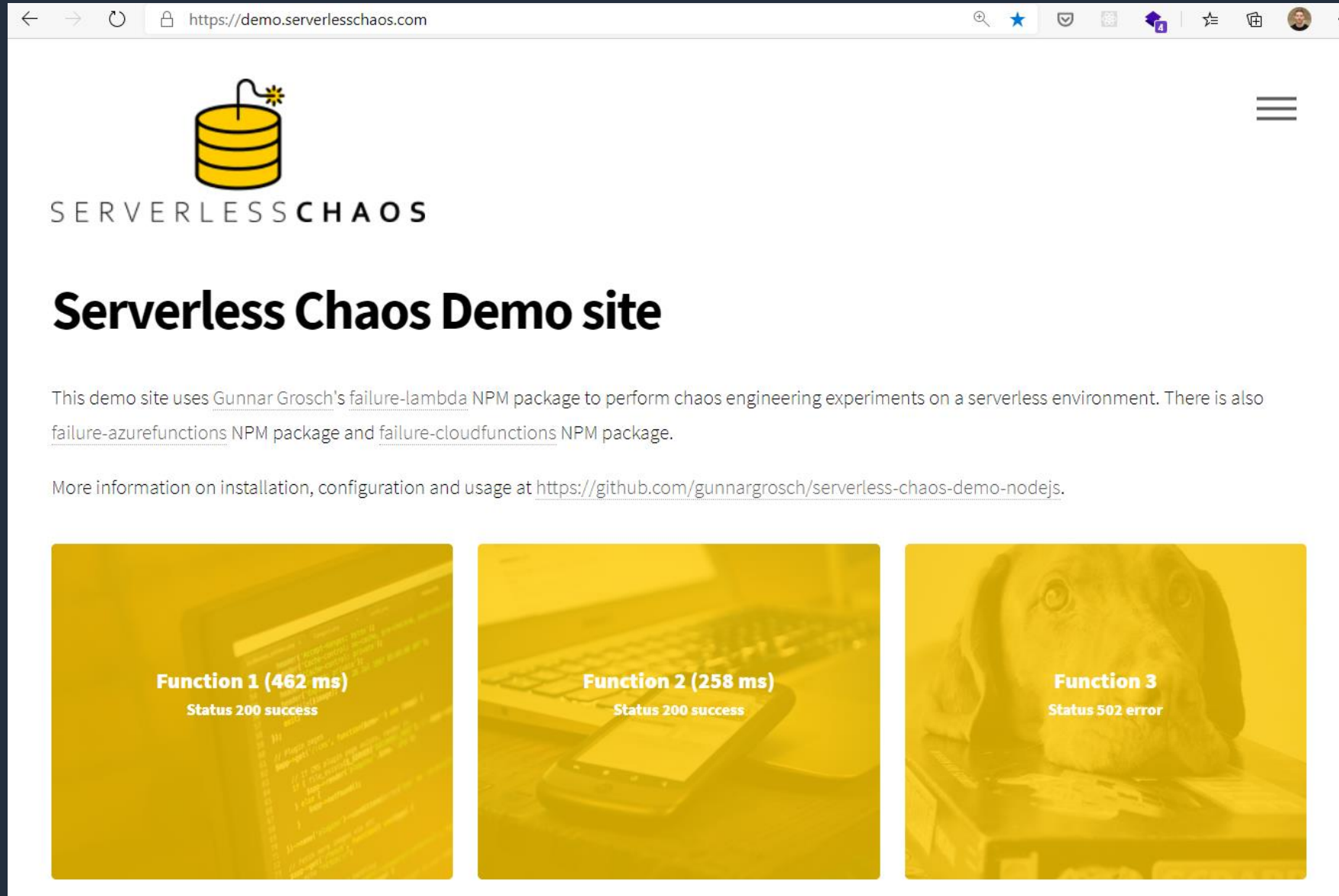
Condition: Inject downstream service failure

Verify: Pass or fail




Verification demo

Verification demo



The screenshot shows a web browser at the URL <https://demo.serverlesschaos.com>. The page features a logo of a yellow database cylinder with a bomb on top, followed by the text "SERVERLESSCHAOS". Below this is the heading "Serverless Chaos Demo site". A paragraph explains that the demo uses [Gunnar Grosch's failure-lambda](#) NPM package for chaos engineering experiments on a serverless environment, and also mentions [failure-azurefunctions](#) and [failure-cloudfunctions](#) NPM packages. A link to <https://github.com/gunnargrosch/serverless-chaos-demo-nodejs> provides more information on installation, configuration, and usage. At the bottom, three yellow-tinted cards display function execution results: "Function 1 (462 ms) Status 200 success" over a code editor background, "Function 2 (258 ms) Status 200 success" over a calculator and smartphone background, and "Function 3 Status 502 error" over a dog's face background.


SERVERLESSCHAOS

Serverless Chaos Demo site

This demo site uses [Gunnar Grosch's failure-lambda](#) NPM package to perform chaos engineering experiments on a serverless environment. There is also [failure-azurefunctions](#) NPM package and [failure-cloudfunctions](#) NPM package.

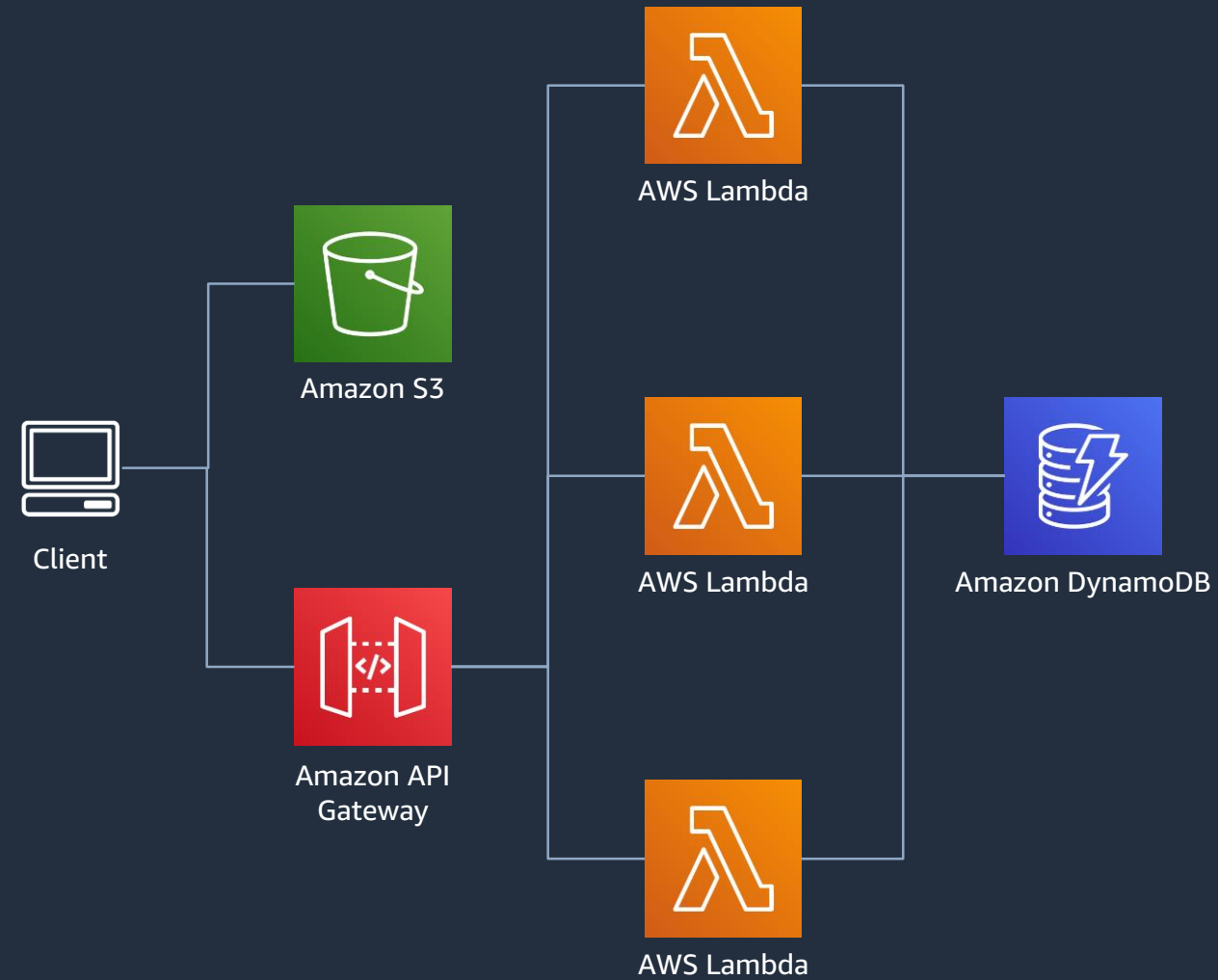
More information on installation, configuration and usage at <https://github.com/gunnargrosch/serverless-chaos-demo-nodejs>.

Function 1 (462 ms)
Status 200 success

Function 2 (258 ms)
Status 200 success

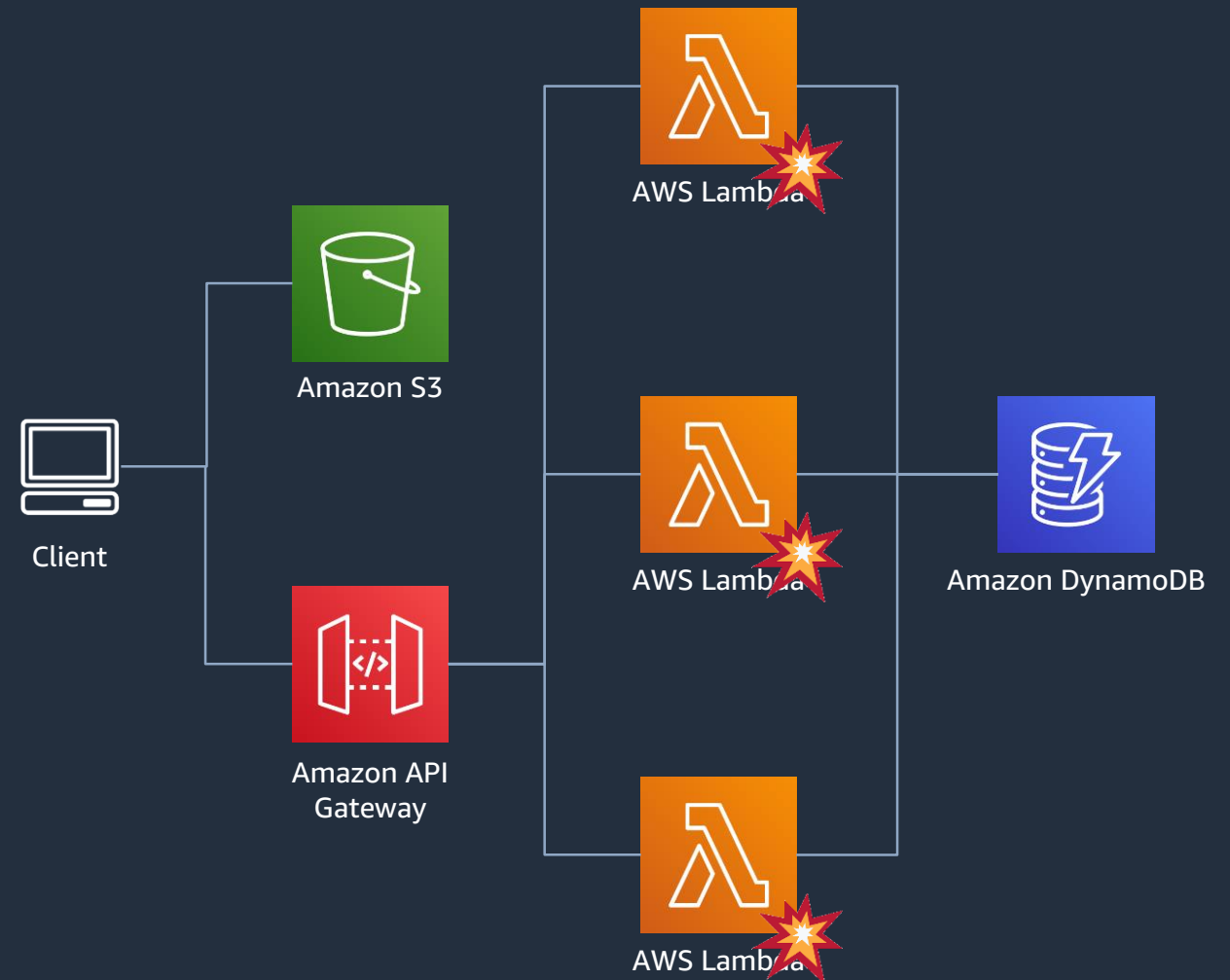
Function 3
Status 502 error

Verification demo



Verification demo

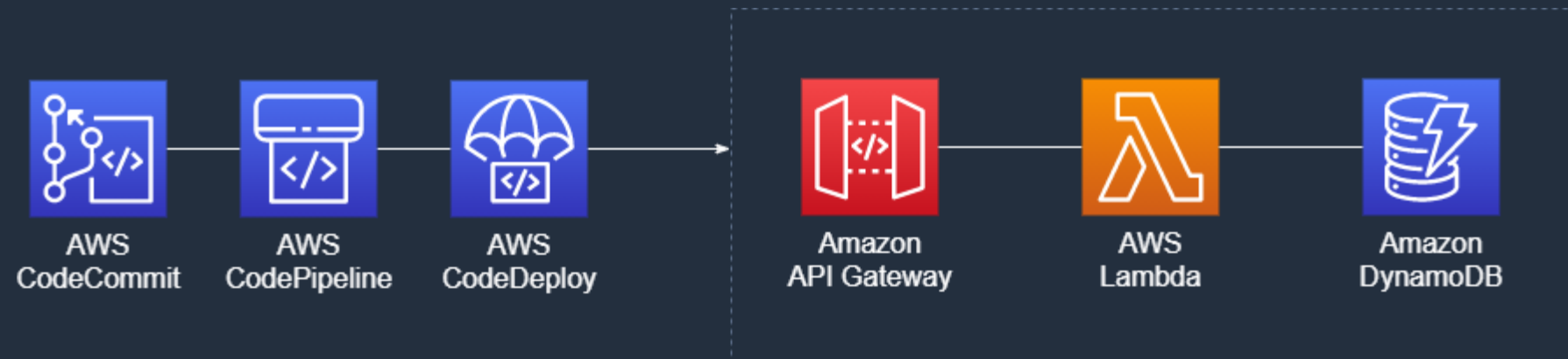
- Condition: Add 100-400 ms latency for each invocation
- Condition: Return error codes on some invocations
- Condition: Intercept and deny connections to DynamoDB



Demo

Making verification continuous

Making verification continuous



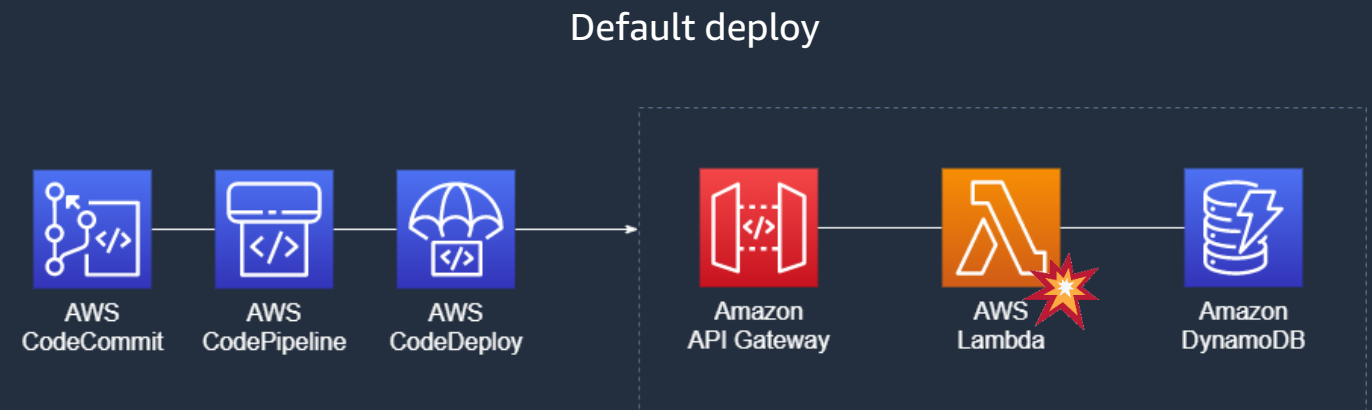
Making verification continuous

Objective: My purchase API should respond in less than 400 ms

Measure: 400 ms threshold

Condition: 100-200 ms latency injection to function

Verify: Pass



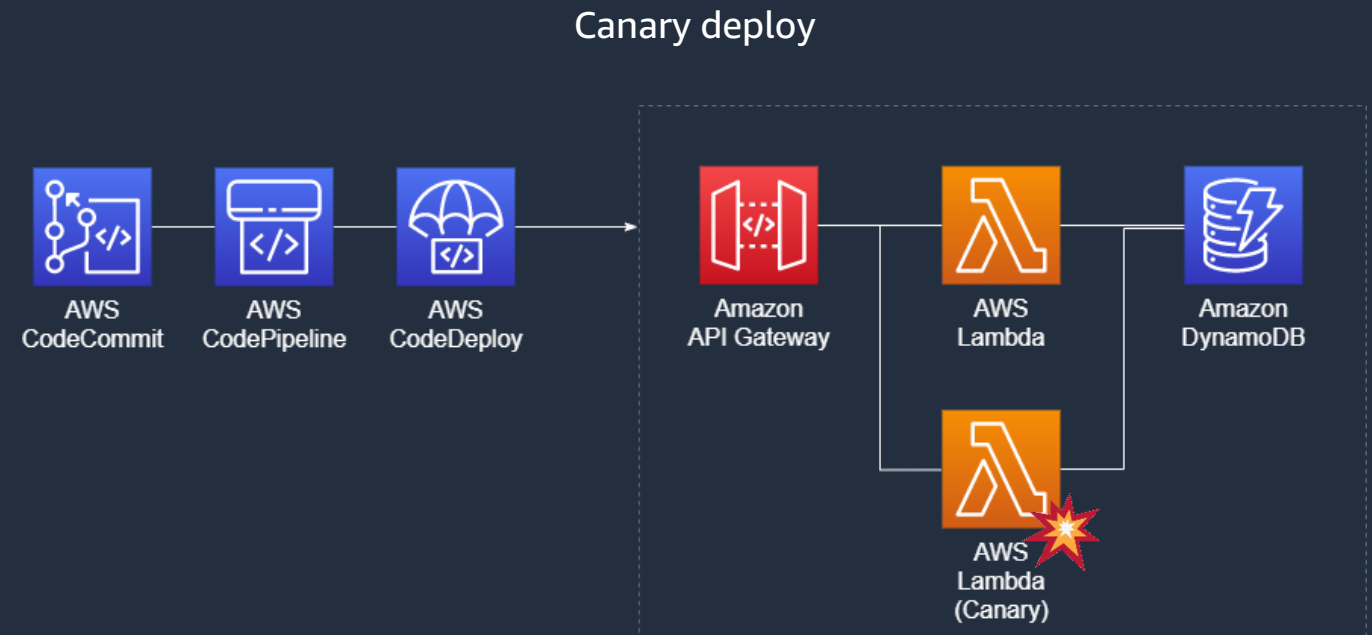
Making verification continuous

Objective: My purchase API should respond in less than 400 ms

Measure: 400 ms threshold

Condition: 100-200 ms latency injection to function

Verify: Fail



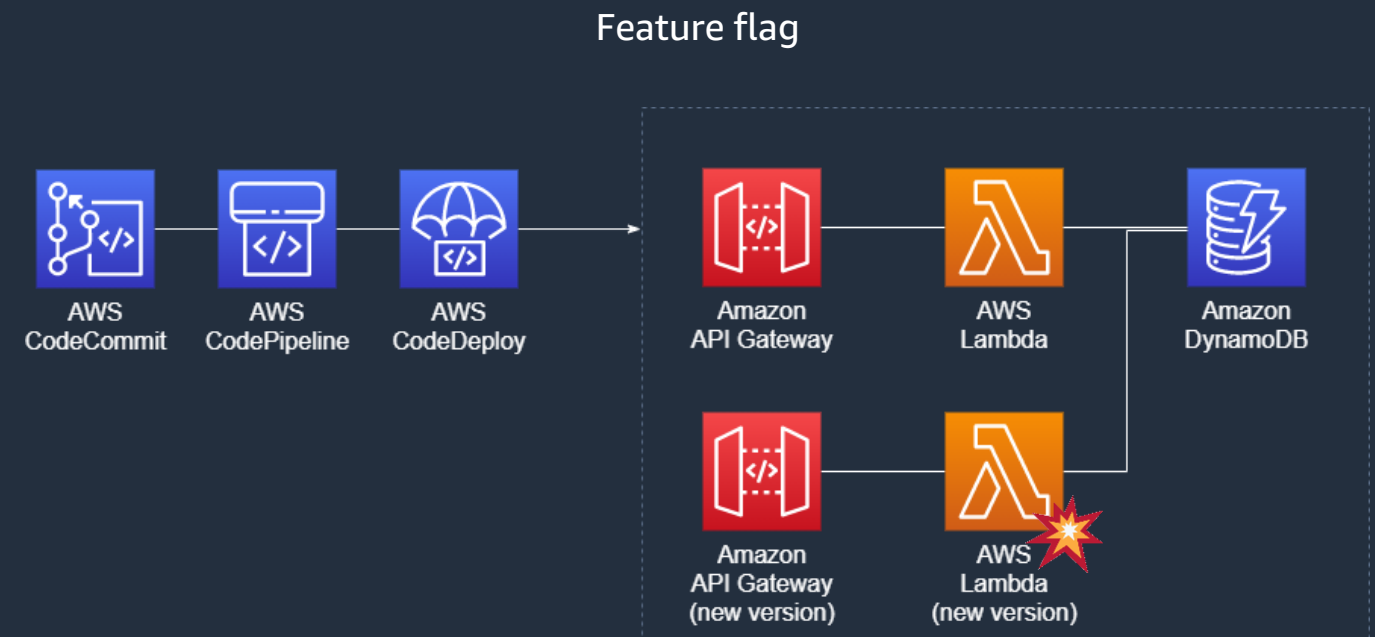
Making verification continuous

Objective: My purchase API should respond in less than 400 ms

Measure: 400 ms threshold

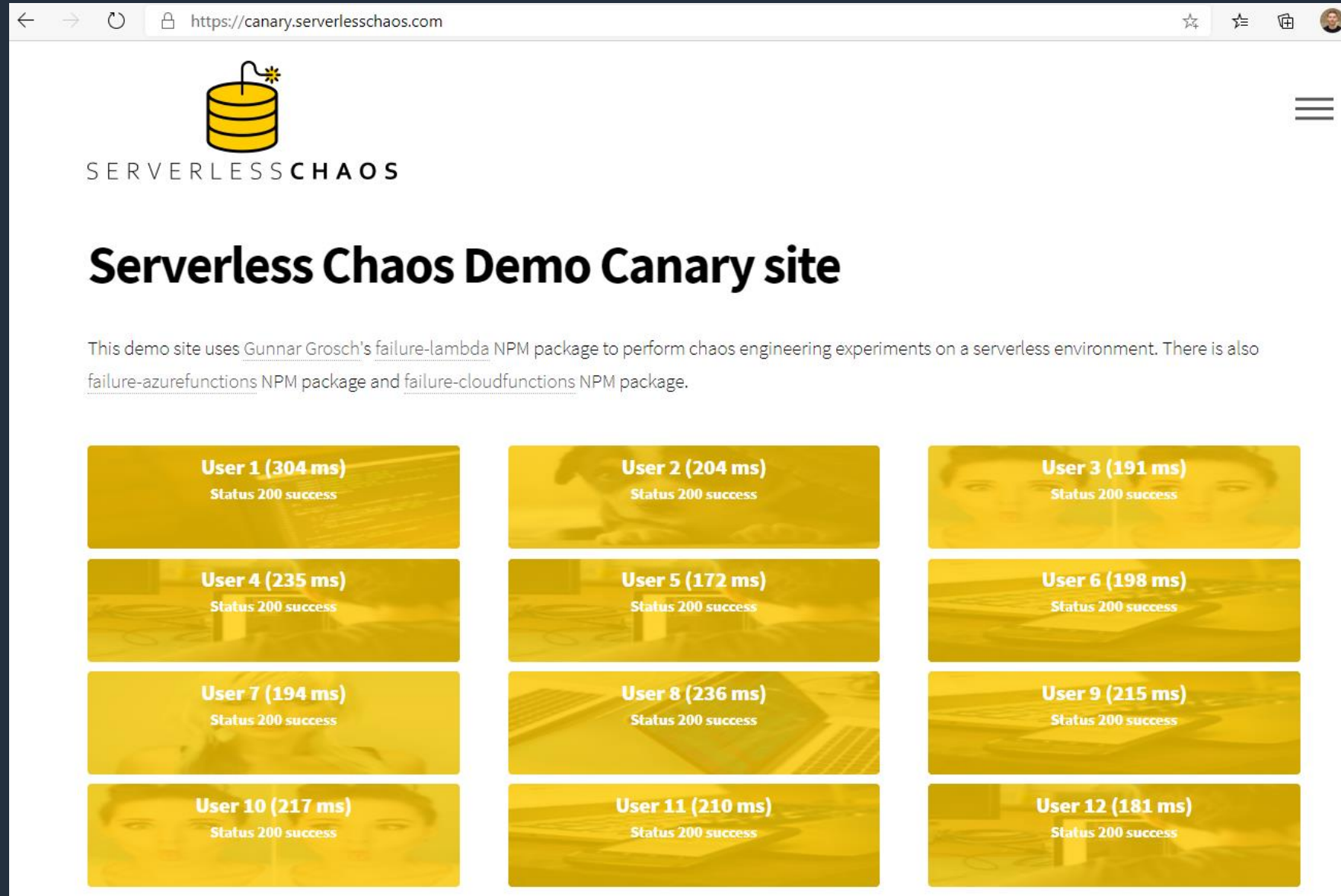
Condition: 100-200 ms latency injection to function

Verify: Pass



Continuous verification demo

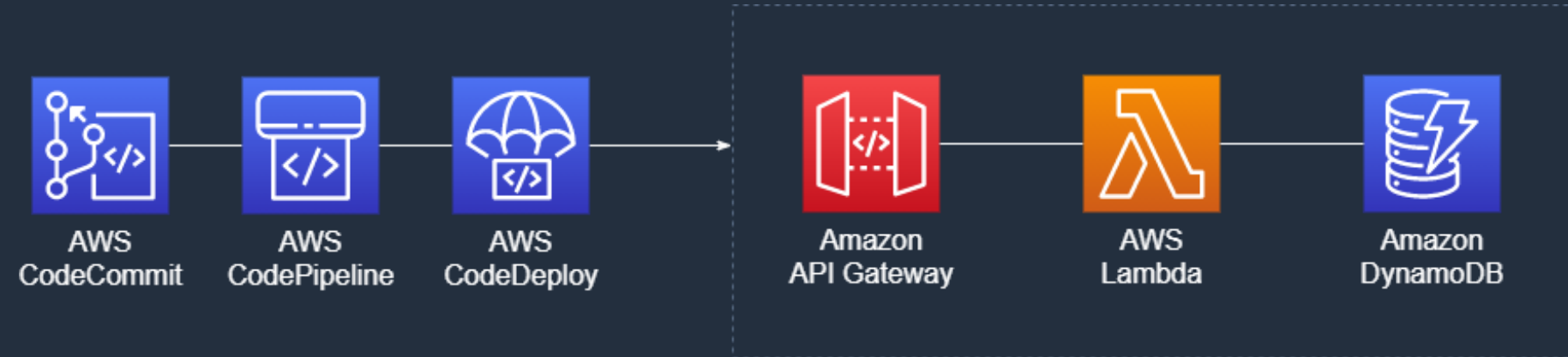
Continuous verification demo



The screenshot shows a web browser window with the URL `https://canary.serverlesschaos.com`. The page features the **SERVERLESS CHAOS** logo, which includes a yellow bomb icon. The main heading is **Serverless Chaos Demo Canary site**. Below the heading, a paragraph explains that the demo site uses [Gunnar Grosch's failure-lambda](#) NPM package for chaos engineering experiments on a serverless environment, and also mentions [failure-azurefunctions](#) and [failure-cloudfunctions](#) NPM packages. The page displays a grid of 12 yellow boxes, each representing a user's request. Each box contains the user number, response time in milliseconds, and the status (all are 'Status 200 success').

User	Response Time (ms)	Status
User 1	304	success
User 2	204	success
User 3	191	success
User 4	235	success
User 5	172	success
User 6	198	success
User 7	194	success
User 8	236	success
User 9	215	success
User 10	217	success
User 11	210	success
User 12	181	success

Continuous verification demo



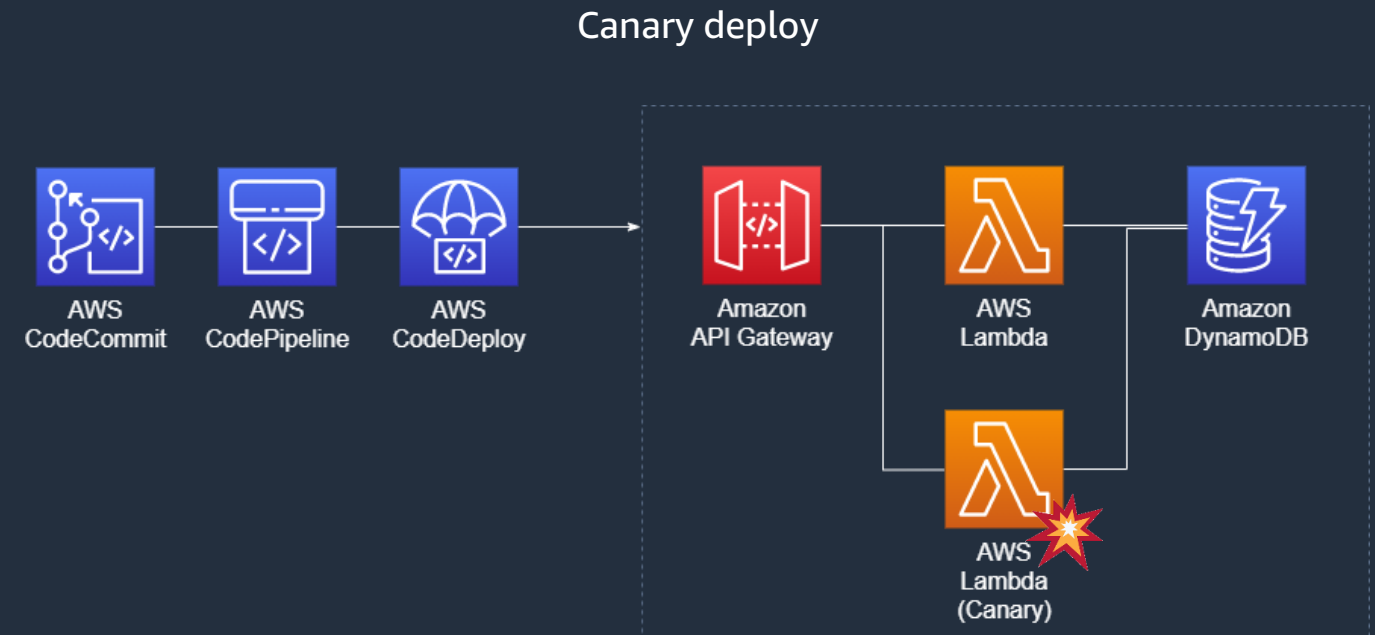
Continuous verification demo

Objective: Users should frequently get new images

Measure: 10 images loaded per minute

Condition: Return error codes on invocations

Verify: Pass or Fail



Demo

Key learnings

Verification tells you why it's important to explore a set of conditions

It's not about breaking things, it's about learning and building confidence

It's easy to get started.

“Testing to ensure that you can meet your availability goals is the only way you can have confidence that you will meet those goals”

Reliability Pillar
AWS Well-Architected Framework

Do you want more?

Reliability pillar

<https://docs.aws.amazon.com/wellarchitected/latest/reliability-pillar/welcome.html>

Serverless Chaos Demo app

<https://demo.serverlesschaos.com>

Failure-lambda

<https://github.com/gunnargrosch/failure-lambda>

Chaos-lambda

<https://github.com/adhorn/aws-lambda-chaos-injection/>



Thank you!

Gunnar Grosch
@gunnargrosch

