Performing chaos in a serverless world

Failover Conf 2020-04-21

Gunnar Grosch
@gunnargrosch

OPSIO

The principles of chaos engineering have been battletested for years using traditional infrastructure and containerized microservices.
But how do they work with serverless functions and managed services?

Agenda

What is chaos engineering?

Motivations behind chaos engineering

Running chaos experiments

The gist of serverless

Challenges with serverless

Serverless chaos experiments

About me

Evangelist and cofounder Opsio

Background in development, operations, and management

Organizer of user groups and conferences

Advocate for serverless and chaos engineering

AWS Serverless Hero

Father of three

@gunnargrosch

OPSIO

What is chaos engineering?

What is chaos engineering?

Chaos engineering is not about breaking things

What is chaos engineering?

Chaos engineering is not only for production

What is chaos engineering?

Chaos engineering is not only for big streaming companies

“Chaos Engineering is the discipline of experimenting on a system in order to build confidence in the system’s capability to withstand turbulent conditions in production.”

principlesofchaos.org

What is chaos engineering?

Chaos engineering is about performing controlled experiments to inject failures

What is chaos engineering?

Chaos engineering is about finding the weaknesses in a system and fixing them before they break

What is chaos engineering?

Chaos engineering is about building confidence in your system and in your organization

Motivations behind chaos engineering

@gunnargrosch

OPSIO

“Everything fails,
all the time!”

Werner Vogels
CTO, Amazon

Motivations behind chaos engineering

Are your customers getting the experience they should?

Is downtime or issues costing you money?

Are you confident in your monitoring and alerting?

Is your organization ready to handle outages?

Are you learning from incidents?

Motivations behind chaos engineering

Don’t ask what happens *if* a system fails; ask what happens *when* it fails

Running chaos experiments

Running chaos experiments

Define steady state

Running chaos experiments

Form your hypothesis

Plan and run your experiment

Running chaos experiments

Measure and learn

Running chaos experiments

Scale up or abort and fix

The gist of serverless

The gist of serverless

"Serverless allows you to build and run applications and services without thinking about servers"

The gist of serverless

No server management

Flexible scaling

Pay for value

Automated high availability

“Serverless is not a technology, it’s a mindset”

Paul Johnston

@gunnargrosch

OPSIO

Challenges with serverless

“There are still servers in serverless”

Countless people

@gunnargrosch

OPSIO

Challenges with serverless

No servers to manage

Less heavy lifting

Lots of services to choose from

Per function and service configuration

More granular architectures

Challenges with serverless

Chaos engineering is a perfect fit for serverless

Serverless chaos experiments

@gunnargrosch

OPSIO

Serverless chaos experiments

Error handling

Timeout values

Events

Fallbacks

Failovers

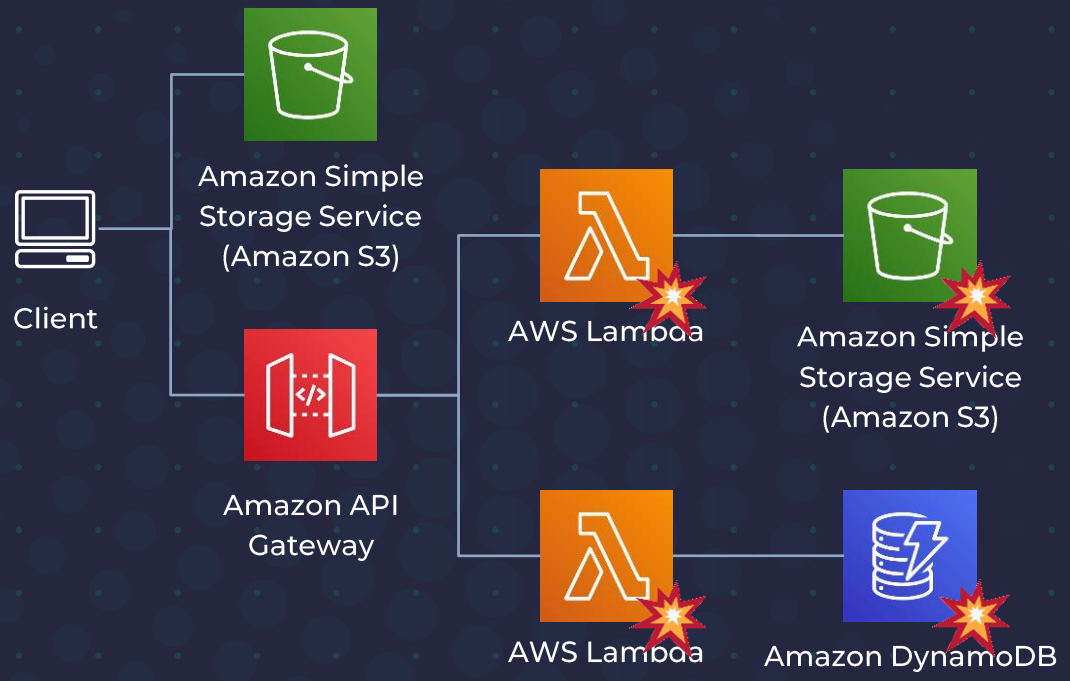
Serverless chaos experiments

Inject errors into your code

Remove downstream services

Alter the concurrency of functions

Restrict the capacity of tables



Serverless chaos experiments

Security policy errors

CORS configuration errors

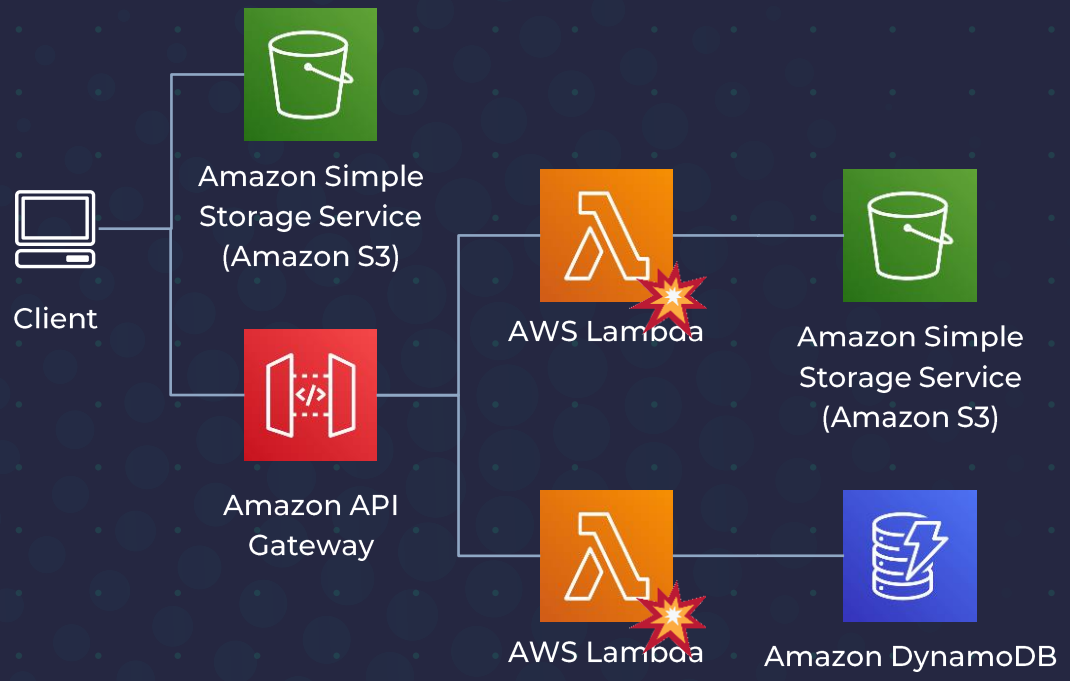
Service configuration errors

Function disk space failure

Serverless chaos experiments

Add latency to your functions

- Cold starts
- Cloud provider issues
- Runtime or code issues
- Integration issues
- Timeouts



Notable chaos engineering tools

Gremlin Alfi
gremlin.com

Chaos Toolkit
chaostoolkit.org

Thundra
thundra.io

Chaos-lambda
Python

Failure-lambda
NodeJS

Failure-azurefunctions
NodeJS

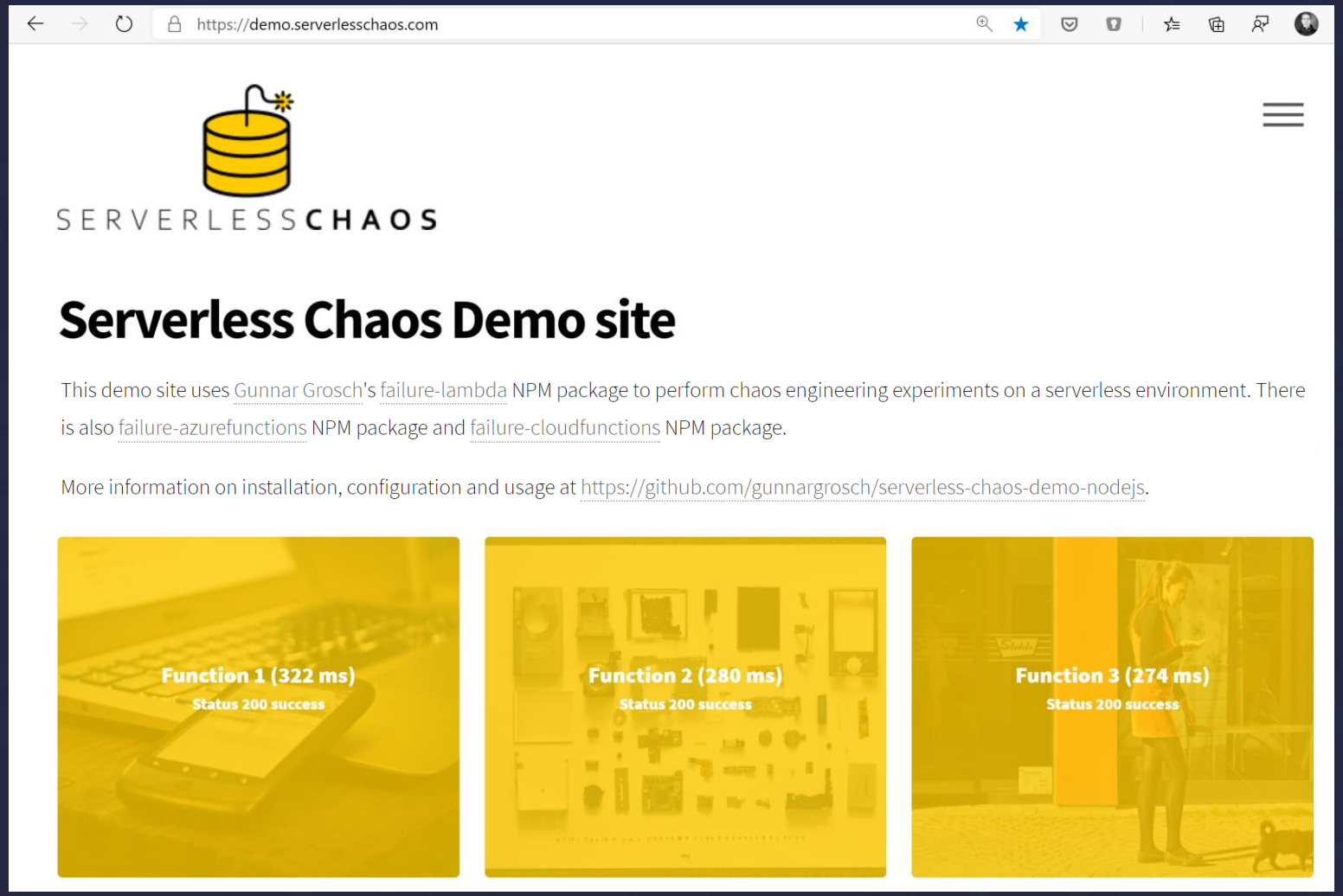
Failure-cloudfunctions
NodeJS

Serverless chaos demo

@gunnargrosch

OPSIO

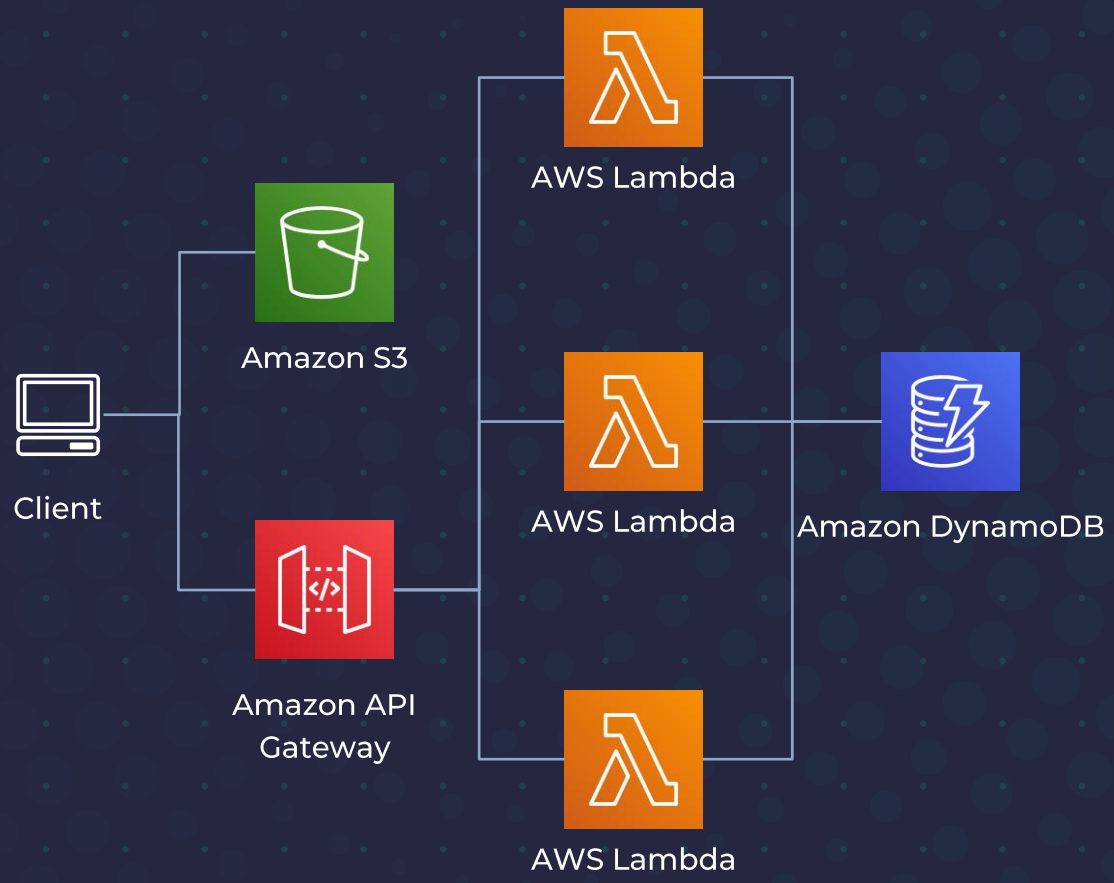
Serverless chaos demo



@gunnargrosch

OPSIO

Serverless chaos demo



Serverless chaos demo

Failure-lambda NPM package

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

Configuration using Parameter Store

Several failure modes

- Latency
- Status code
- Exception
- Disk space
- Blacklist

```
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,
  "blacklist": [
    "s3.*.amazonaws.com",
    "dynamodb.*.amazonaws.com"
  ]
}
```

Serverless chaos demo

What if my function takes an extra 300 ms for each invocation?

What if my function returns an error code?

What if I can't get data from DynamoDB?

Hypothesis: If we inject failure to functions then my application will use graceful degradation.

Demo

Recap

Everything fails, all the time

Serverless doesn’t make your application resilient

Chaos engineering helps us find weaknesses and fix them

Chaos engineering is about building confidence

Chaos engineering is a perfect fit for serverless

It’s not rocket science; you can do it!

Do you want more?

Follow @serverlesschaos on Twitter

Serverless Chaos Demo app: https://demo.serverlesschaos.com

Failure-lambda: https://github.com/gunnargrosch/failure-lambda

Failure-cloudfunctions: https://github.com/gunnargrosch/failure-cloudfunctions

Failure-azurefunctions: https://github.com/gunnargrosch/failure-azurefunctions

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

Serverless chaos lab: https://github.com/jpbarto/serverless-chaos-lab

YouTube videos and repositories: https://grosch.se

Thank you!

#qa-gunnar-grosch

Gunnar Grosch
@gunnargrosch