Releasing improved serverless functions with confidence

Tutorial at O'Reilly Software Architecture
November 05, 2019, Berlin

Gero Vermaas & Jochem Schulenklopper
Xebia, The Netherlands

Abstract from O'Reilly Software Architecture

Jochem Schulenklopper and Gero Vermaas get you started by describing serverless architectures, serverless applications, AWS Lambda functions, and the serverless framework to specify and deploy such applications. Right from the start, you'll fork an application and deploy it as a serverless application to a production environment. They also describe the Scientist library and its concept as described and used by GitHub in refactoring a core Git library, as well as the backstory of Scientist, how GitHub used it in testing an improved implementation, and some (sometimes surprising) results of applying its "scientific method" for things running in production.

They compare the approach used by Scientist ("testing candidates in production, comparing their outcomes with a control") with other ways to verify the correctness and quality of software implementations and architecture designs, including unit and component testing, integration testing, performance and load testing, production simulation, A/B testing, blue-green deployments, feature flags, canary deployments, and architecture fitness functions, and make the case that the scientific method of testing improvements in production really adds value in guaranteeing whether a (refactored) implementation objectively and conclusively performs better.

You'll apply this "release improved code to production with confidence" approach on the small serverless application that already runs in production and make improvements to the code (choosing from AWS Lambda functions in Java, Python, Node.js, Go, or Ruby programming languages), deploy the improved code to production, and compare that with the original baseline applications during ongoing production traffic. Based on the outcomes, you can confidently conclude whether your changes are actual improvements over the current implementation.

Today's tutorial addresses...

Briefly,

- What are serverless applications?
- What is AWS Lambda? What is Serverless Framework?
- How can I deploy a serverless function to AWS Lambda?

Main topics,

- How can I compare an alternative implementation with an already running function in production?
- How can a set up a structured experiment validating functionality and performance characteristics?

Outline for today's tutorial

[15 min.] Brief introduction into serverless architectures, AWS Lambda functions, Serverless Framework

[5 min.] Hands-on: accessing today's online environment, starting a track in Instruqt

[10 min.] Brief description of different approaches for verifying software

[20 min.] Description of Scientist library as used by GitHub in refactoring some service, and a comparable library for testing serverless applications

[105-120 min.] Hands-on: applying Scientist in testing and releasing improvements in a sample serverless application **(challenges 1 - 7)**

Serverless applications, two definitions

"Backend as a Service" / BaaS: applications using ecosystem of cloud-accessible services: databases, authentication, messaging, analytics

"Functions as a Service" / FaaS: applications composed of small, event-triggered, ephemeral functions, fully managed by a third party

Common features: no resource management by consumer, granular computing resources, automatic load-based scalability, usage-based pricing, high availability

Common use cases: event data stream or multimedia processing, HTTP REST APIs powering web and mobile applications, CI/CD pipelines, machine learning

AWS Lambda - https://aws.amazon.com/lambda/

AWS Lambda is an event-driven, serverless computing platform provided by Amazon as a part of the Amazon Web Services. It is a computing service that runs code in response to events and automatically manages the computing resources required by that code

The purpose of Lambda is to help building smaller, on-demand applications that are responsive to events. Lambda functions can be coded in Node.js, Python, Java, Go, Ruby, C# and other languages

Competitors: Google Cloud Functions, Azure Functions, ...

Serverless Framework

Serverless Framework is a set of developer tools to develop and deploy cloud applications at multiple FaaS providers

In this tutorial we'll use Serverless Framework to easily deploy serverless applications to AWS: AWS Lambda functions and some other AWS resources

You will invoke the **`sls`** command and edit a **`serverless.yml`** configuration file, but no more detailed knowledge of the Serverless Framework is required

See https://serverless.com/ for more information

The three serverless functions in this tutorial

There's a range of use cases for serverless applications, on multiple dimensions

- *functionality*: plain data retrieval, lookup, compute, processing
- *state*: functions without data storage, with some persistent data store
- *character*: simple, or part of complex setup with additional (platform) services
- *interface*: plain text, JSON objects, images, audio, video

Today we keep things simple, to focus on *improving* three serverless functions:

- HTTP GET https://endpoint/round - for rounding floating point numbers
- HTTP POST https://endpoint/sort - for sorting simple JSON lists
- HTTP GET https://endpoint/fizzbuzz - for generating FizzBuzz sequences

And what about this "Instruqt" thing?

instruqt

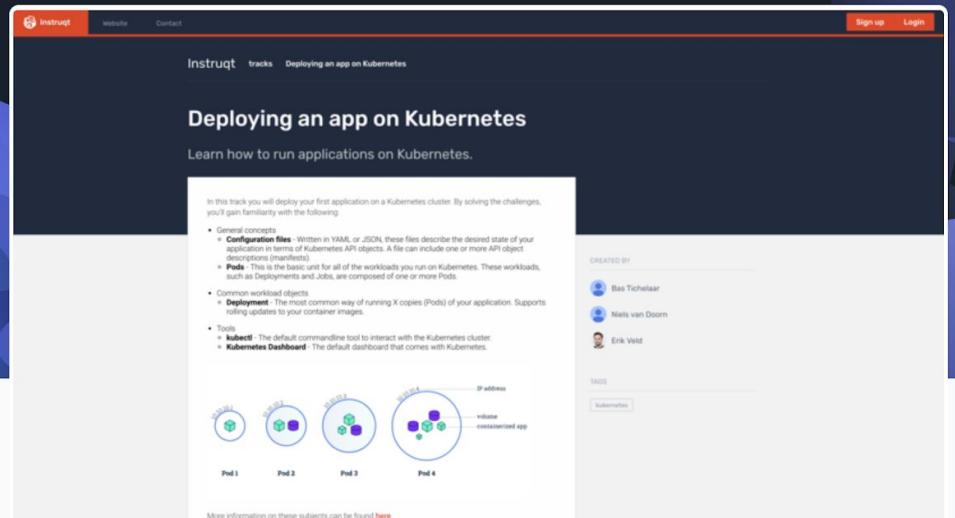
Product Use cases Pricing Blog Company Go to platform

MEET INSTRUQT

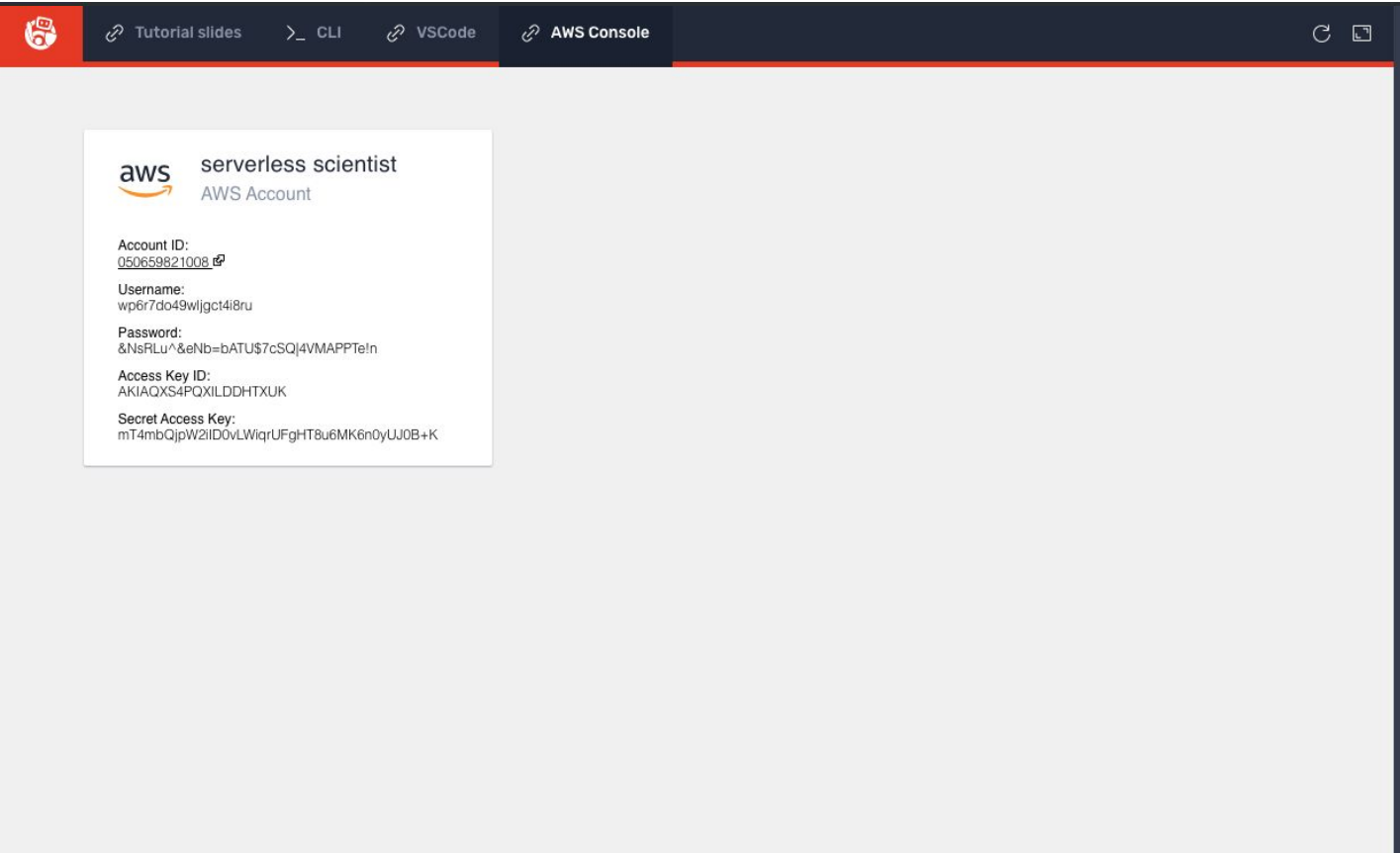
The online interactive IT learning platform

Create sandbox environments using real infrastructure to train your teams, customers & prospects. Define challenges to guide users and uncover the true value of your product.

Play existing content Create content



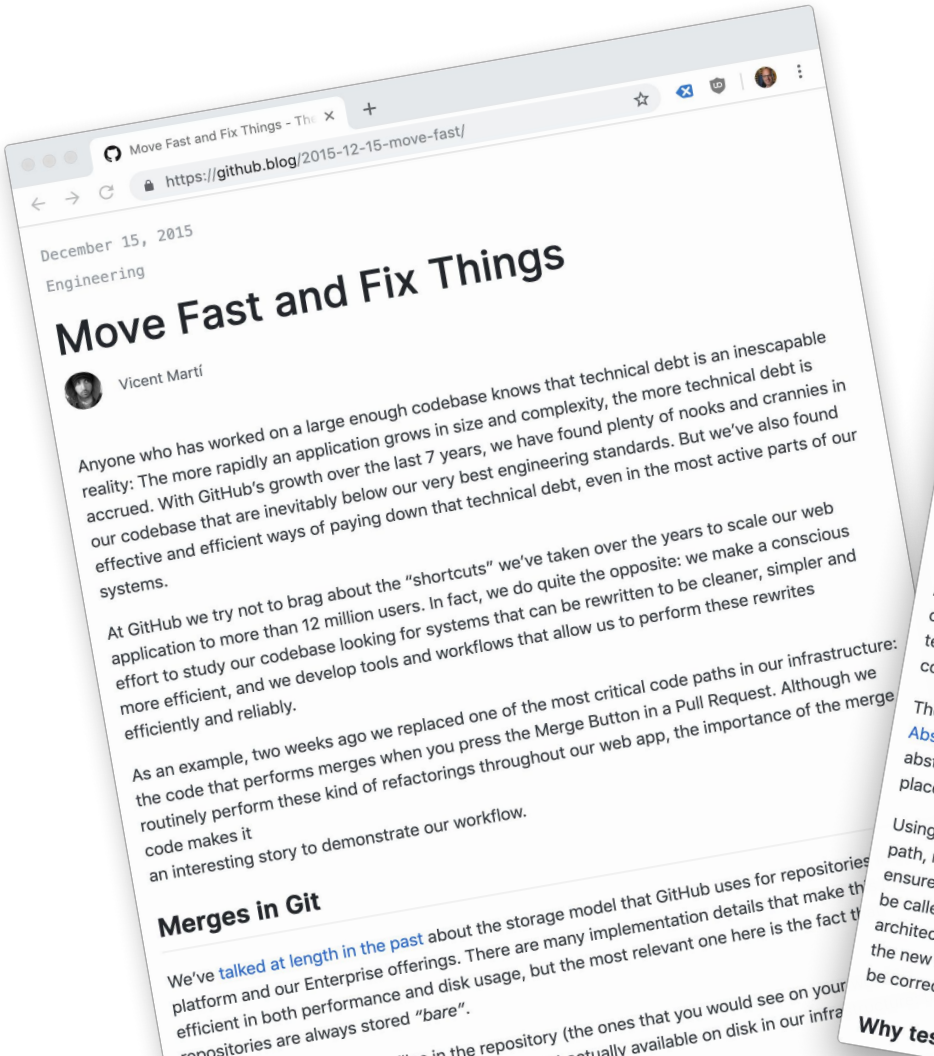
Tabs in Instruqt



Hands-on: accessing today's online environment in Instruqt


Tests in production
QA
Tests not in production

QA method	Test against	Phase	How to get test data
Unit testing	Test spec	Dev	Manual
Integration testing	Test spec	Dev	Manual
Performance testing	Test spec	Tst	Dump production traffic /simulation
Acceptance testing	User spec	Acc	Manual
Feature flags	User expectations	Prd	Segment of production traffic
A/B-testing	Comparing options	Prd	Segment of production traffic
Blue/green deployments	User expectations	Prd	All production traffic
Canary releases	User expectations	Prd	Early segment of production traffic


Move Fast and Fix Things - Th
https://github.blog/2015-12-15-move-fast/
December 15, 2015
Engineering
Move Fast and Fix Things
Vicent Martí
Anyone who has worked on a large enough codebase knows that technical debt is an inescapable reality: The more rapidly an application grows in size and complexity, the more technical debt is accrued. With GitHub's growth over the last 7 years, we have found plenty of nooks and crannies in our codebase that are inevitably below our very best engineering standards. But we've also found effective and efficient ways of paying down that technical debt, even in the most active parts of our systems.
At GitHub we try not to brag about the "shortcuts" we've taken over the years to scale our web application to more than 12 million users. In fact, we do quite the opposite: we make a conscious effort to study our codebase looking for systems that can be rewritten to be cleaner, simpler and more efficient, and we develop tools and workflows that allow us to perform these rewrites efficiently and reliably.
As an example, two weeks ago we replaced one of the most critical code paths in our infrastructure: the code that performs merges when you press the Merge Button in a Pull Request. Although we routinely perform these kind of refactorings throughout our web app, the importance of the merge code makes it an interesting story to demonstrate our workflow.
Merges in Git
We've talked at length in the past about the storage model that GitHub uses for repositories
platform and our Enterprise offerings. There are many implementation details that make th
efficient in both performance and disk usage, but the most relevant one here is the fact t
repositories are always stored "bare".


Scientist: Measure Twice, Cut
https://github.blog/2016-02-03-scientist/
February 3, 2016
Engineering
Scientist: Measure Twice, Cut Over Once
Jesse Toth
Today we're releasing Scientist 1.0 to help you rewrite critical code with confidence.
As codebases mature and requirements change, it is inevitable that you will need to replace or rewrite a part of your system. At GitHub, we've been lucky to have many systems that have scaled far beyond their original design, but eventually there comes a point when performance or extensibility break down and we have to rewrite or replace a large component of our application.
Problem
A few years ago when we were faced with the task of rewriting one of the most critical systems in our application — the permissions code that controls access and membership to repositories, teams, and organizations — we began looking for a way to make such a large change and have confidence in its correctness.
There is a fairly common architectural pattern for making large-scale changes known as Branch by Abstraction. It works by inserting an abstraction layer around the code you plan to change. The abstraction simply delegates to the existing code to begin with. Once you have the new code in place, you can flip a switch in the abstraction to begin substituting the new code for the old.
Using abstractions in this way is a great way to create a chokepoi
path, making it easy to switch over to the new code when the t
ensure that the behavior of the new system will match the old s
be called in all places where the old system was called. For suc
architecture, this pattern only fulfilled half of the requirements. W
the new system would be used in all places that the old system was, but also that it
be correct and match what the old system did.
Why tests aren't en

GitHub

Proposal: new software QA method, "Scientist"

Situation:

- We have an existing software component running in production: "control"
- We have an alternative (and hopefully better) implementation: "candidate"

Questions:

- Is the candidate **behaving correctly** (or just as control) in all cases? (functionality)
- Is the candidate **performing qualitatively better** than the control? (response time, stability, memory use, resource usage stability, ...)

Requirements for such a Scientist in software

Ability to

- Experiment: test controls and (multiple) candidates with production traffic
- Observe: compare results of controls and candidates

Additionally, for practical reasons in performing experiments

- Easily route traffic to single or multiple candidates
- Increase sample size once more confident of candidates
- No impact for end-consumer
- **No change required in control**
- No persistent effect from candidates in production

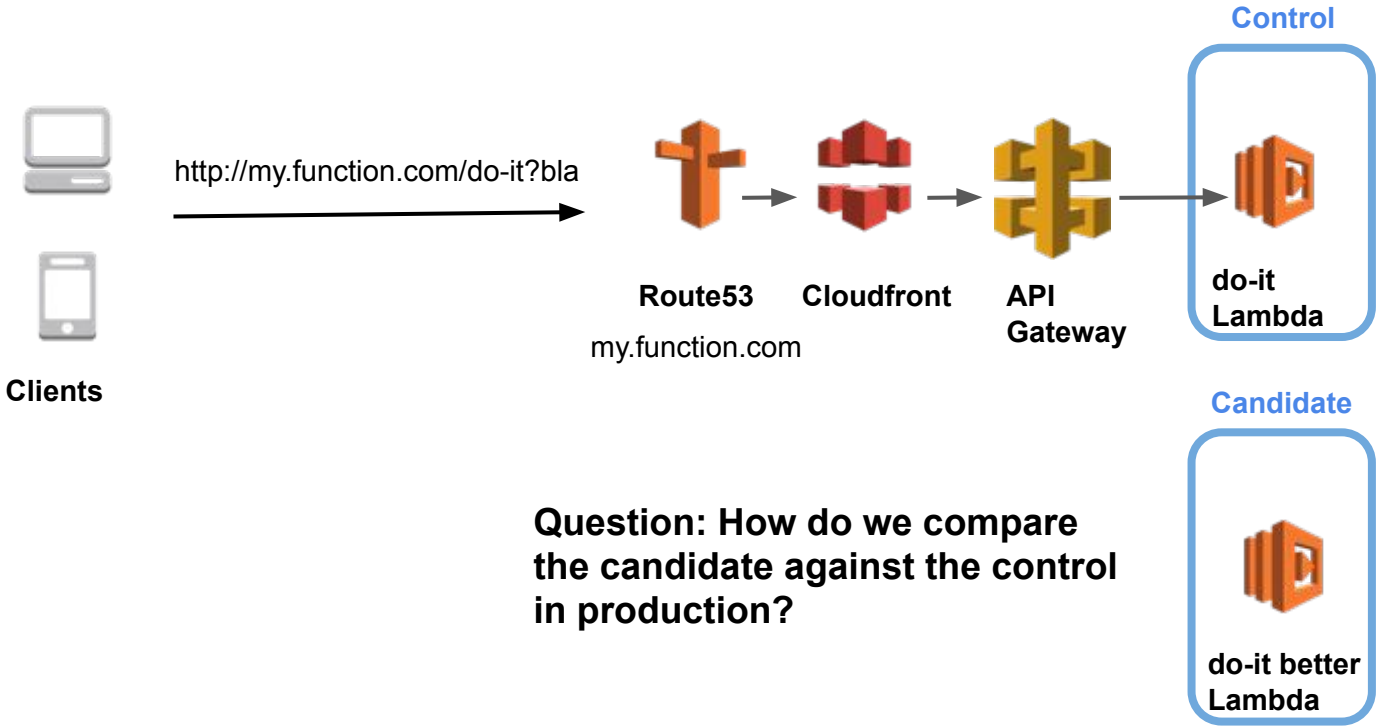
Extra requirements for a *serverless* Scientist

No impact on clients

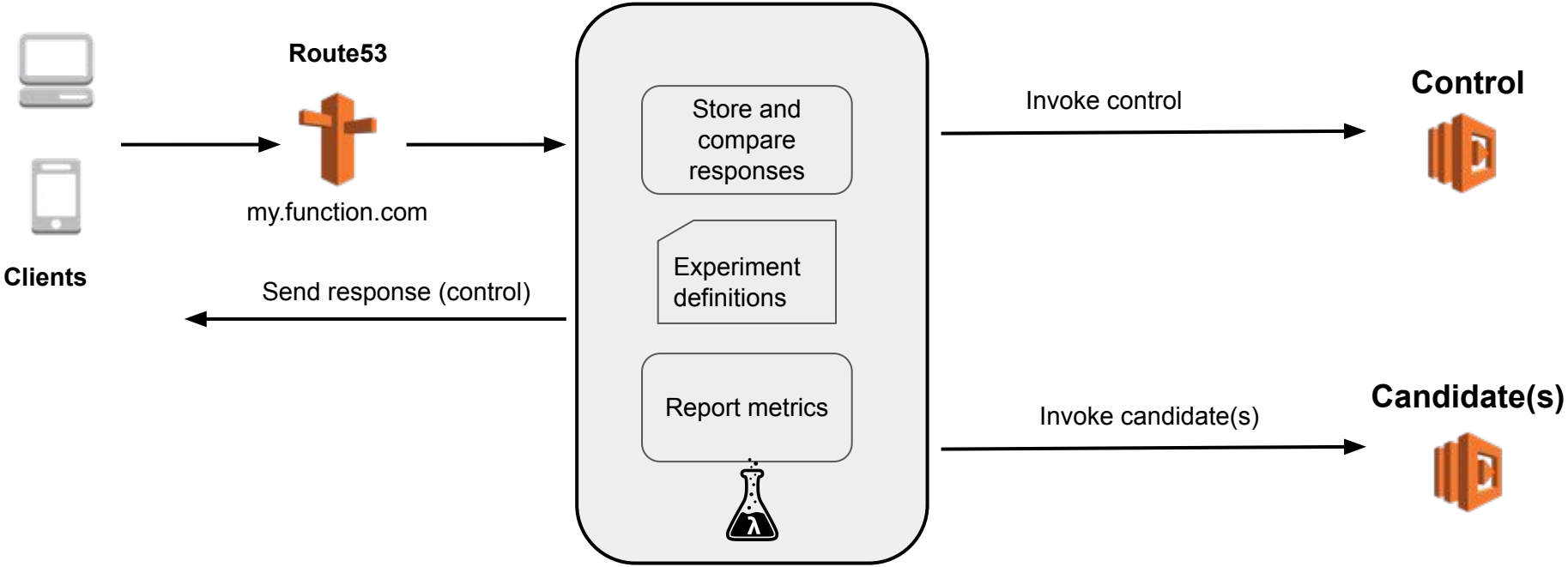
Instant insight
in results

Minimal operations effort

Typical setup for serverless functions on AWS



Setup of Serverless Scientist



When to apply a Scientist in software validation?

- Refactoring an existing function running in production
- Changing the implementation language of a function
- Changing an internal dependency (library) for another
- Replacing custom logic by an external dependency, an external service
- Changing a function's operating environment (e.g., language runtime upgrade)
- Changing available resources for a function (e.g., vCPUs, RAM, I/O)

When is a Scientist approach less applicable?

- When the interface of a candidate isn't backwards compatible with a control's *(Explanation: the production requests to controls are duplicated to the candidate, but then resulting in differences in the responses)*

Hands-on: testing and releasing improved serverless applications (challenges 1-7)

Challenge 1: Get the code for today's tutorial

The steps for this challenge (it's not a real "challenge" yet):

- Press the green “Start” button on the “Refactor serverless functions with confidence” page
- Fork the repository for today's tutorial following the instructions
- Use the “Check” button to validate if all is OK

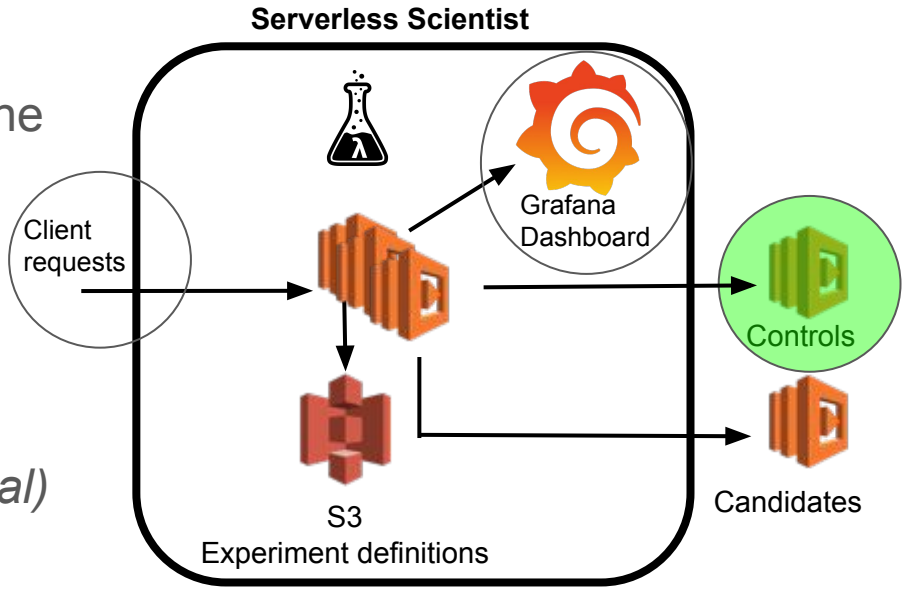
Challenge 2: Deploy and use a simple function

We've already prepared a simple function for rounding floating point numbers. Now we need to deploy that function to production as the control.

The steps in Instruqt:

- Deploy the production version, a.k.a. the control

N.B. This will also deploy the other controls which we will use later in the tutorial)

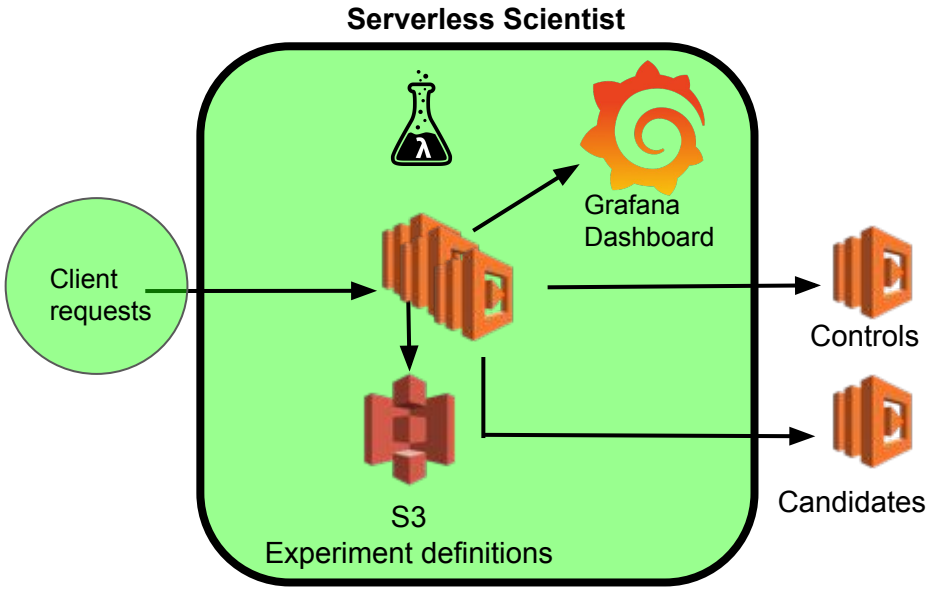


Challenge 3: Set up the Serverless Scientist

We have some serverless functions ("controls") running in production.
Next, we need to install the Serverless Scientist library for the next challenges.

The steps in Instruqt:

- Deploy the Serverless Scientist library to AWS
- Upload the experiment definition file used for this tutorial
- Validate that the control function is accessible via the experiment

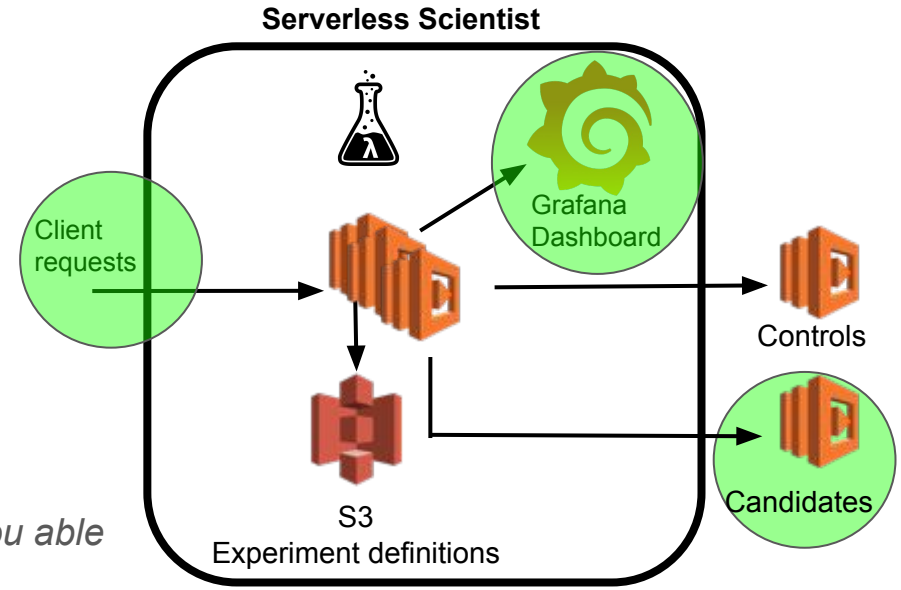


Challenge 4: Deploy, test a candidate for round()

An existing implementation for round() is running in production as control.
We now deploy an existing alternative implementation as a candidate.

The steps in Instruqt:

- Navigate to the candidate for round()
- Deploy the candidate function
- Check the Scientist dashboard whether the candidate is better than the control



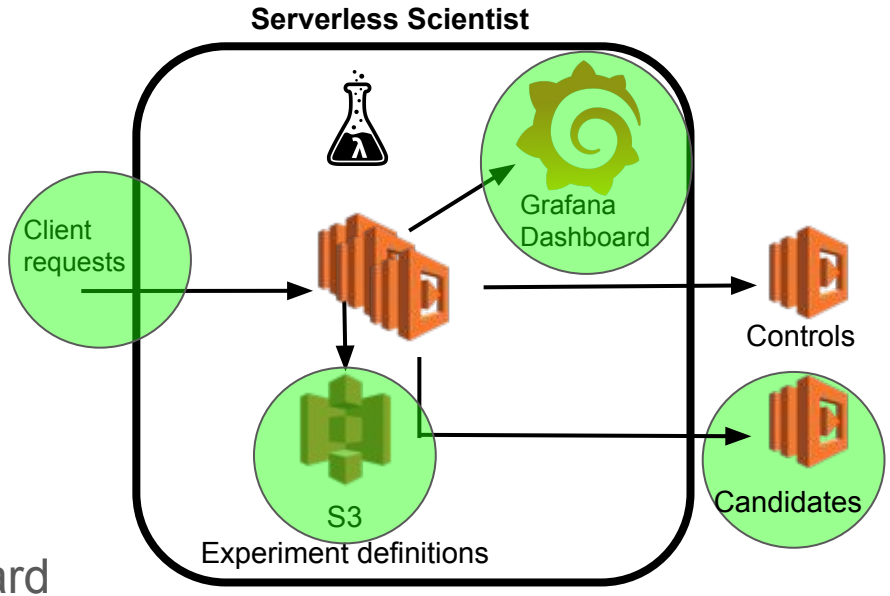
NB: There's a small peculiarity in the candidate code. Does production traffic trigger that behavior and are you able to see the difference with the control?

Challenge 5: Fix, deploy, test candidate for sort()

A sort() function is running in production as control.
We should complete and deploy a candidate function, and check its correctness.

The steps in Instruqt:

- Find the (already coded) candidate for a sort function
- Fix two minor things so that it'll be working correctly
- Deploy the now completed candidate (The experiment already includes this candidate if you left the name intact)
- Check that it's working on the dashboard

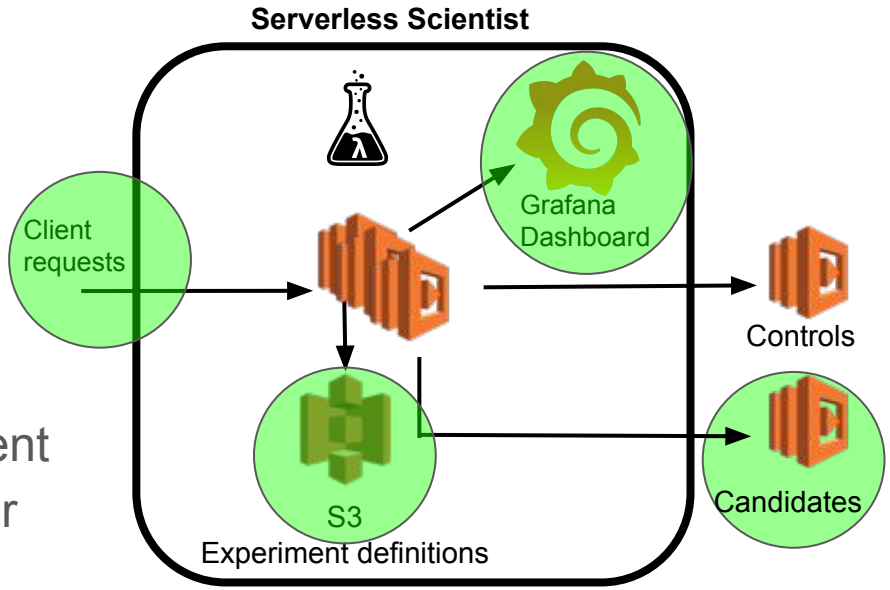


Challenge 6: Create and test fizzbuzz() function

A FizzBuzz function is running in production as control, but you can do better...
(especially since we put in a minor bug - will you detect it in an experiment?)

The steps in Instruqt:

- Take a code skeleton in a language of your choice
- Develop the fizzbuzz() function as a candidate for FizzBuzz in production
- Deploy your candidate
- Include your candidate in the experiment
- Check whether your candidate is better than the control on the dashboard



Requirements for HTTP GET endpoint `/fizzbuzz`

Input: plain HTTP GET request with two optional URL query string parameters

- `from` (a positive integer), otherwise start with `1`
- `to` (an integer), otherwise end at `from` + 99

Output: plain text list of Fizz Buzz numbers increasing from `from` up to `to`

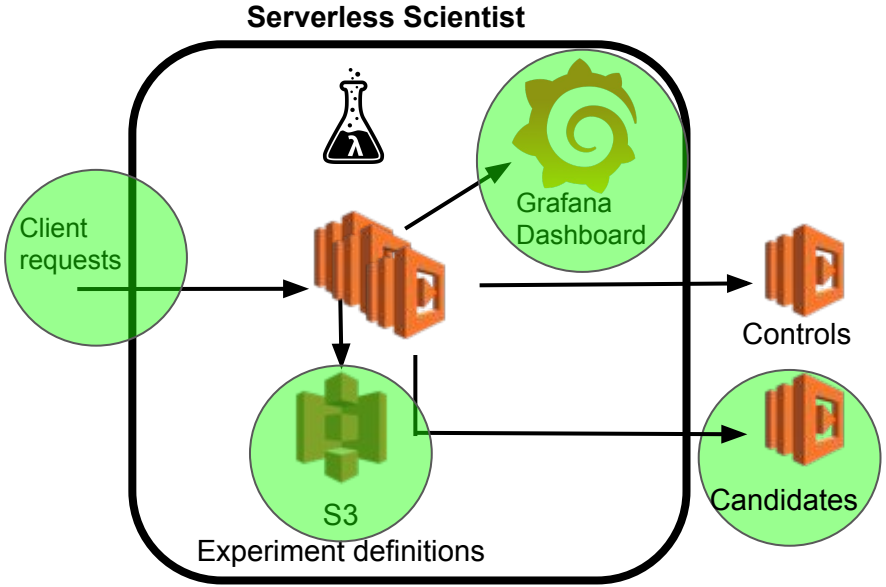
- FizzBuzz numbers separated by `\n`, but no trailing `\n`
- “Fizz” if divisible by 3,
- “Buzz” if divisible by 5,
- “Fizz Buzz” if divisible by 15
- number otherwise
- if `from` higher than `to` then output nothing at all

Challenge 7: Change a function's configuration

A CPU-intensive function is running in production as control, deriving a key from a password (PBKDF2, for "password-based key derivation function"). Change its configuration (its allocated memory size) and notice the impact on performance.

The steps in Instruqt:

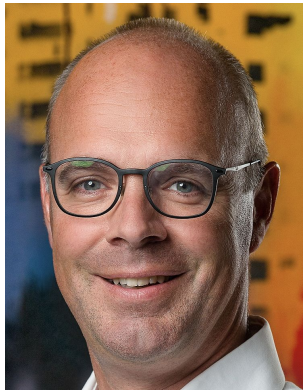
- Define and deploy candidates with different memory allocations
- Notice the performance differences on direct requests
- Notice the performance differences on the experiments' dashboard



Thanks for your participation in this tutorial

Feel free to contact us:

- Gero Vermaas, gvermaas@xebia.com, @gerove
- Jochem Schulenklopper, jschulenklopper@xebia.com, @jschulenklopper
- Xebia: https://xebia.com/contact, Instruqt: https://instruqt.com/contact/



Xebia

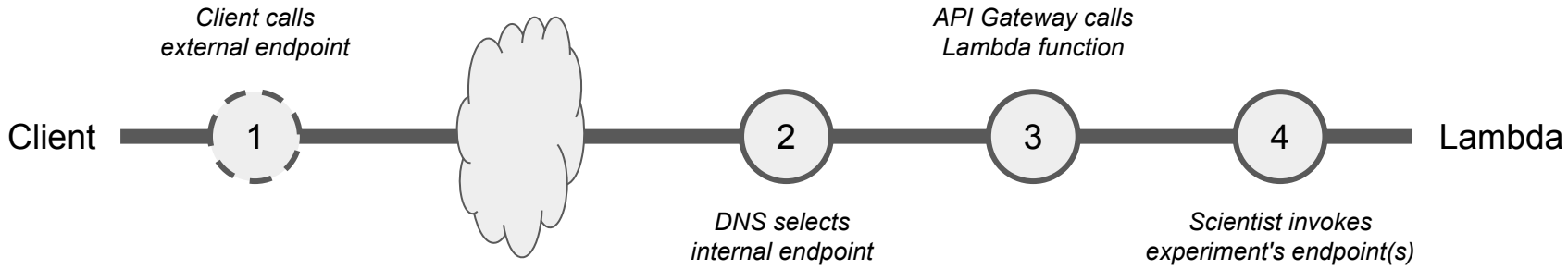
instruqt

Uh... and what about promoting that candidate of which we're now confident that it's better than the control?

From a client's request to a Lambda function

Four major configuration points that determines which Lambda function is called:

1. (Client's request to an API endpoint - client decides which endpoint is called)
2. Proxy or DNS server - routing an external endpoint to an internal endpoint
3. API Gateway configuration - mapping a request to a Lambda function
4. Serverless Scientist - invoking functions for experiment's endpoints



Three options to release candidate as new control

2. Change the route for an external endpoint to another internal endpoint

Direct traffic from old control to new candidate -> becomes new control

3. Change API Gateway configuration: associate other Lambda function

Change the existing production Lambda function to a new implementation: a Lambda function previously a candidate in an experiment

4. Change setup of experiment: inject candidate as new control

Change ARNs of control to the previous candidate in the experiment
(and possibly specify the old control as a new candidate)

</tutorial>

More about that experiment configuration file

```
experiments:
 rounding-float:
   comparators:
     - body:
     - statuscode:
     - headers:
       - content-type
   path: round
   control:
     name: Round Node8.10
     arn: arn:aws:lambda:{AWSREGION}:{AWSACCOUNT_ID}:function:control-round
   candidates:
     candidate-1:
       name: Round Python3-math
       arn: arn:aws:lambda:{AWSREGION}:{AWSACCOUNT_ID}:function:candidate-round-python3-math
     candidate-2:
       name: Round python-3-round
       arn: arn:aws:lambda:{AWSREGION}:{AWSACCOUNT_ID}:function:candidate-round-python3-round
```

https://api.serverlessscientist.com/**round**?number=62.5

On Instruqt

Instruqt, https://instruqt.com/ is an interactive online IT learning platform

It originated from Xebia as an online training tool with gamification, initially deployed at tech conferences for online competitions (using real gaming consoles)

Nowadays Instruqt is being used

- by technology vendors, to showcase product releases by giving developers an opportunity to experience the products in a live setting
- by enterprises, to train employees on new IT products in online sessions
- by individuals, to get hands-on experience with cool and new tech products

On Serverless Framework

Serverless Framework, https://serverless.com/, is a set of tools to define, develop, configure and deploy serverless applications on multiple cloud hosting platforms

Online documentation can be found at https://serverless.com/framework/docs/

Online forum is hosted at https://forum.serverless.com/

Repositories for Serverless Framework, examples and plugins are at https://github.com/serverless/

On AWS Lambda and API Gateway

AWS Lambda - run and scale backend code without provisioning or managing servers

- https://aws.amazon.com/lambda/
- Developer guide: https://docs.aws.amazon.com/lambda/latest/dg/welcome.html
- API reference: https://docs.aws.amazon.com/lambda/latest/dg/API_Reference.html

AWS API Gateway - a managed service to create, publish, maintain, monitor, and secure API endpoints

- https://aws.amazon.com/api-gateway/
- Developer guide: https://docs.aws.amazon.com/apigateway/latest/developerguide/welcome.html
- API reference: https://docs.aws.amazon.com/apigateway/api-reference/

Most AWS services are bound to a region; if AWS resource data isn't visible in the AWS console, make sure that you're looking at the correct region. We're using **eu-west-1** (Ireland) in this tutorial.

On Serverless, AWS Lambda + API Gateway

If the response of a HTTP GET or POST request is similar to

"server IP address could not be found" or **"could not resolve host"**

then the serverless functions haven't been deployed yet - the endpoint is missing.

If the response of a HTTP GET or POST request is

{"message": "Internal server error"}

then the request handler is not attached to the Lambda function, or it contains a syntax error.

If the response of a HTTP GET or POST request is

{"message":"Missing Authentication Token"}

then the URL path (or the HTTP method) is incorrect.

Controls or candidates in Go need to be built with the correct target architecture.
On the CLI in the proper directory:

$ make build; sls deploy

On serverless applications made with Serverless

It might happen that **serverless** could be updated during the tutorial. If so, then

$ npm i serverless

should do the trick. For today, you can skip this update as well; it comes up at unexpected times.

Honestly, local development of functions running online at AWS Lambda and API Gateway can be messy, as not all (AWS) resources are available in your local development environment. Serverless Framework provides some resources in offline mode, but not everything.

Logging to CloudWatch is your friend. CloudWatch includes log items sent to standard output.

Developing and testing core logic without the need of a local environment can also be done at online IDEs like https://repl.it or https://aws.amazon.com/cloud9/.

For mocking AWS services locally, LocalStack, https://github.com/localstack/localstack might be helpful but support for AWS's APIs is incomplete.

Our tutorial runs 'outside' Instruqt as well

But the list of requirements makes it easier to run in from Instruqt:

- Local development environment (IDE, SDK for languages you're using, Git)
- Node.js including npm, for installing and using Serverless Framework
- Serverless Framework installed on your host
- Account at AWS, user within that AWS account with proper authorizations

Steps to set up the environment for doing the challenges locally:

- Clone or download Serverless Scientist:
  `git clone https://gitlab.com/practicalarchitecture/serverless-scientist.git`
- Deploy Serverless Scientist's services and this tutorial's controls and candidates from their respective directories using `serverless deploy`.

On Scientist

GitHub's original Ruby gem can be found at https://github.com/github/scientist

The backstory of GitHub's Scientist can be found at

- "Move Fast and Fix Things", https://github.blog/2015-12-15-move-fast/
- "Scientist: Measure Twice, Cut Over Once", https://github.blog/2016-02-03-scientist/

Implementations / ports of the Scientist gem to other languages are listed at https://github.com/github/scientist#alternatives

On Serverless Scientist

The code repository is at
https://gitlab.com/practicalarchitecture/serverless-scientist

The experiment definition YAML file in **scientist/** specifies the experiment, controls, and candidates used.