



KubeCon



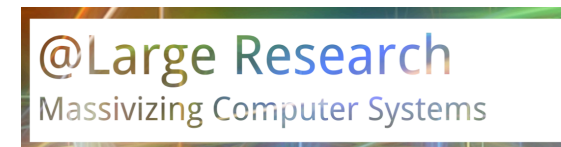
CloudNativeCon

Europe 2019

Serverless Operations

From dev to production

Erwin van Eyk



Why serverless computing?

Operational Model



Minimal operational logic

“Infinite” autoscaling

Built-in tooling: monitoring, tracing, health checking, etc.

Cost Model



Pay for what you use

No upfront/periodic costs

Granular billing

Development Model

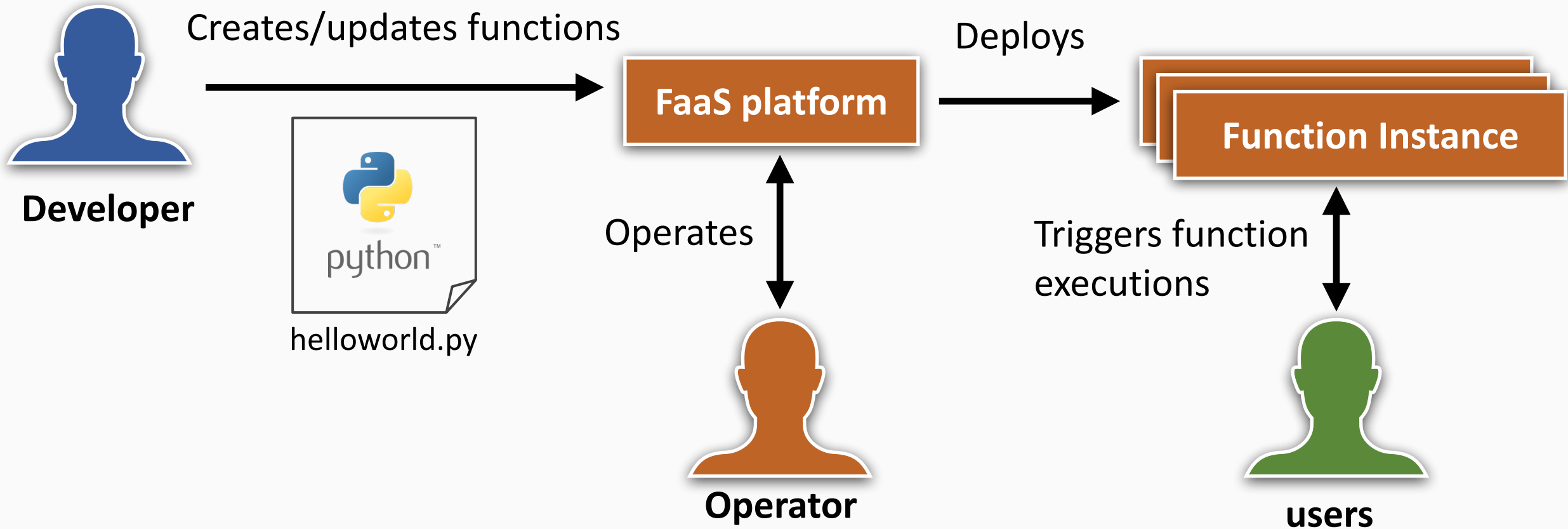


High-level abstractions

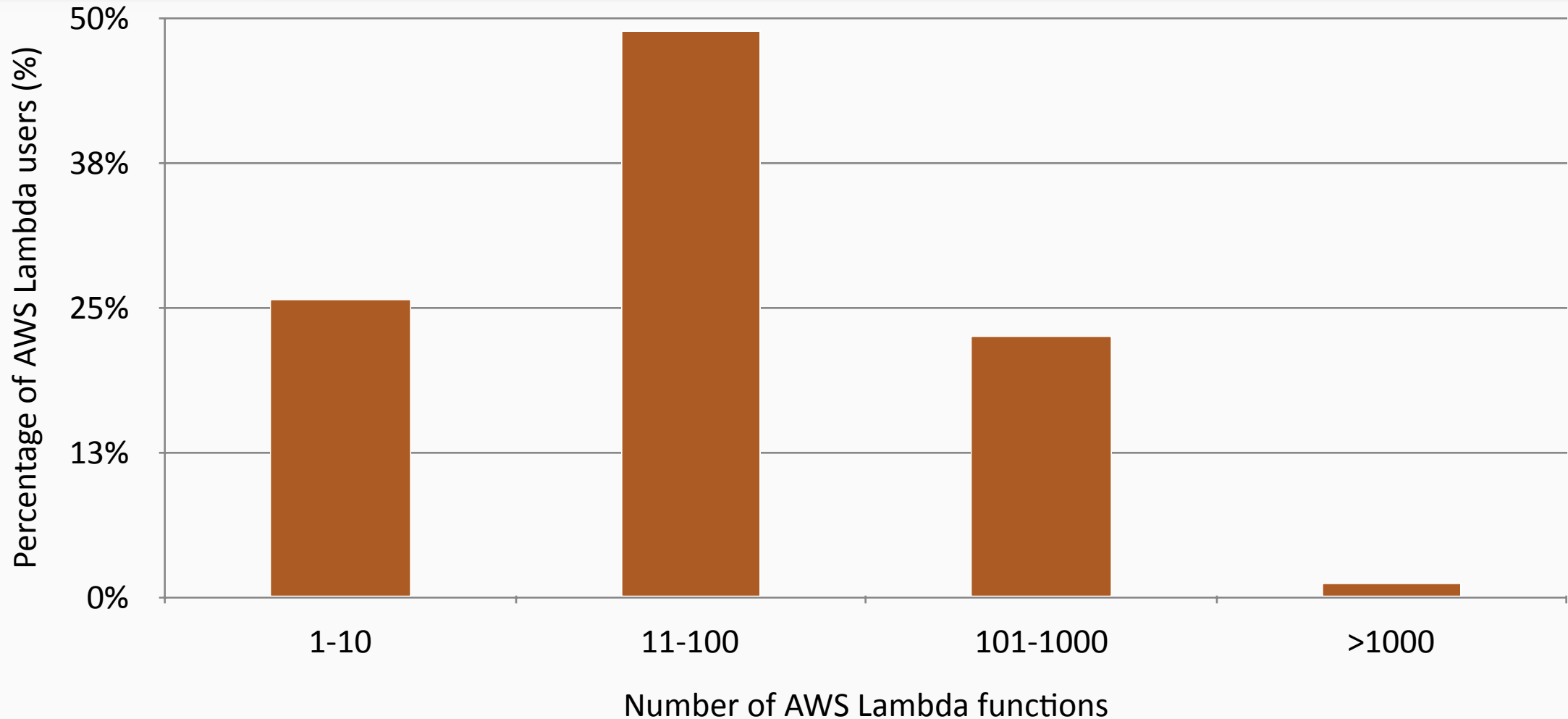
Pre-provided integrations

Language-agnostic

Function-as-a-Service (FaaS) in a nutshell

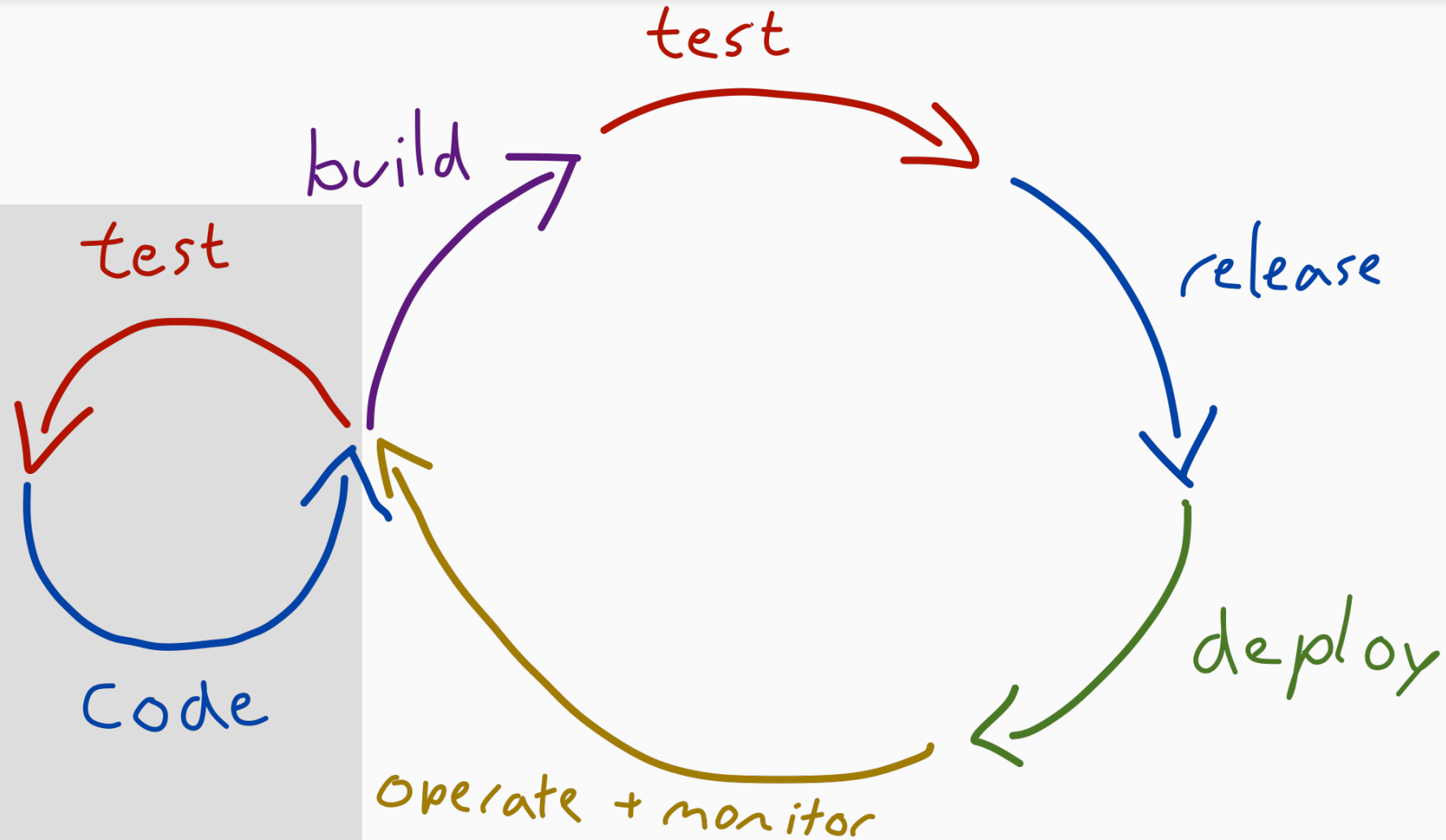


Serverless users use more functions...



...for which we need structured operational processes.

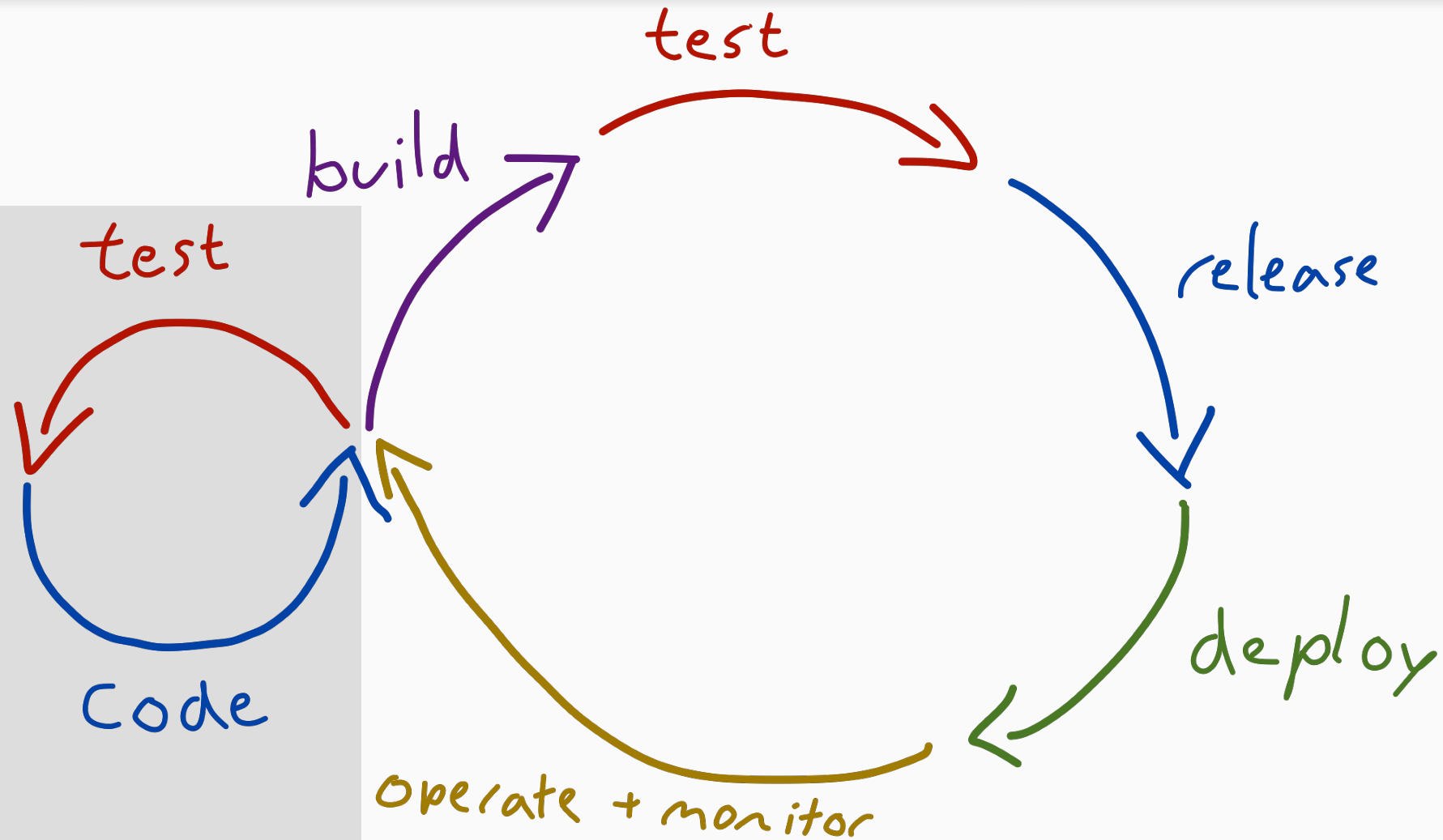
The traditional DevOps lifecycle



Development Loop

Operation Loop

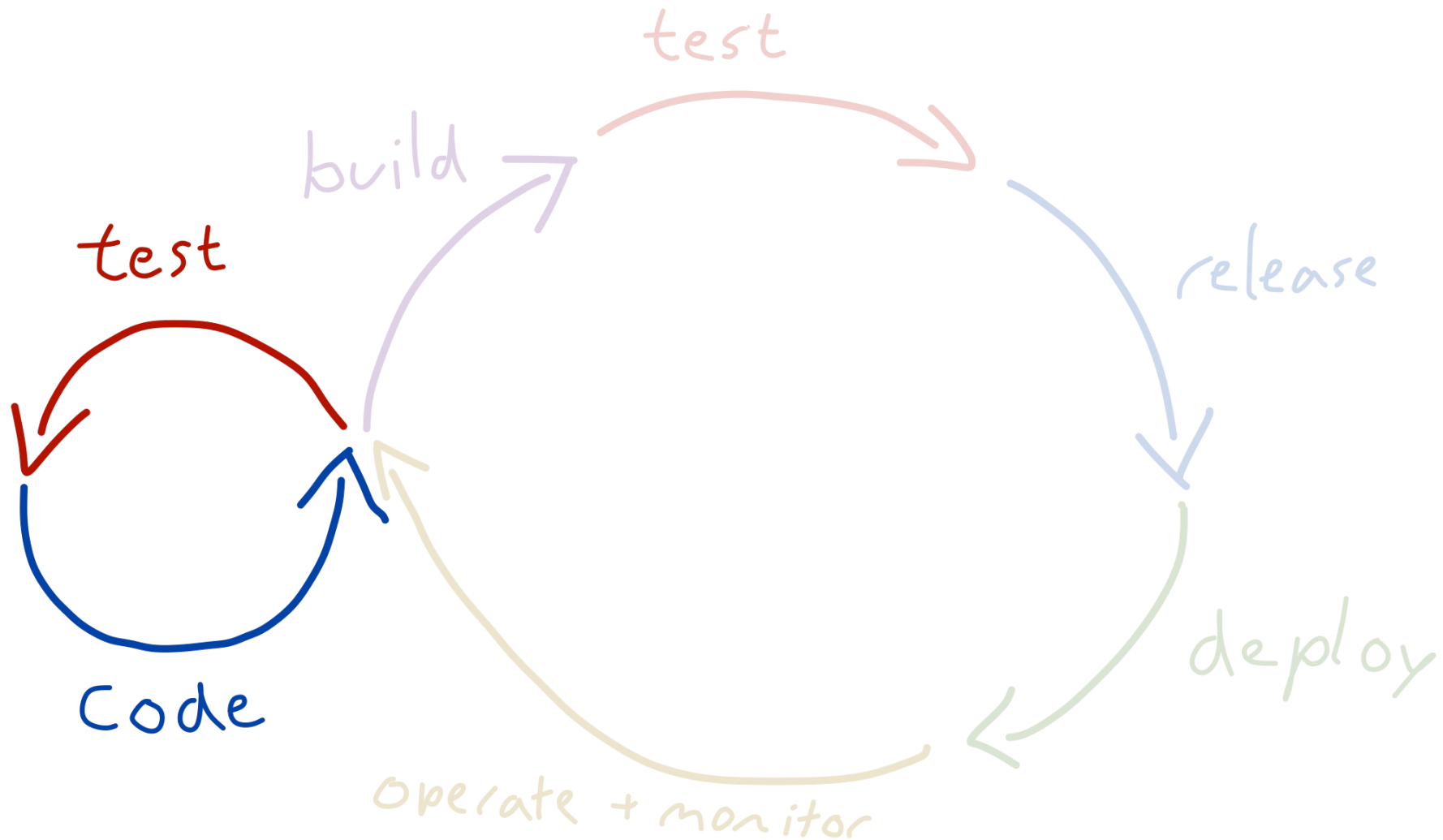
The **serverless** DevOps lifecycle



What are good practices (specifically) for serverless operations?

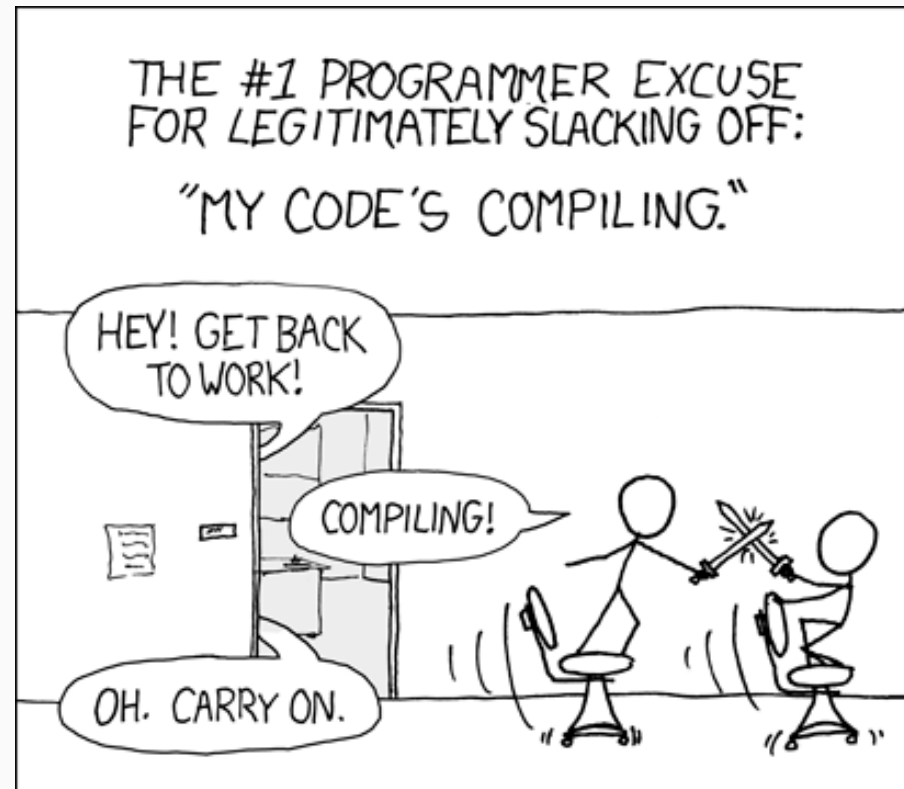
Development Loop

code, test.



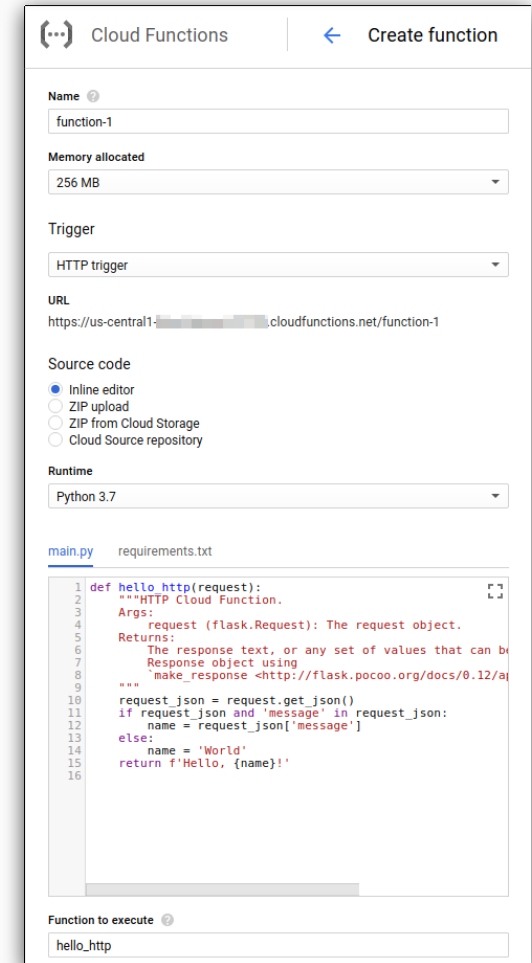
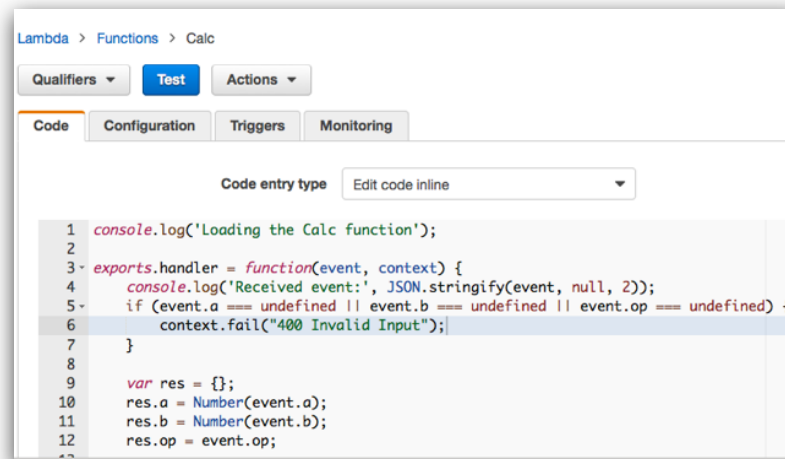
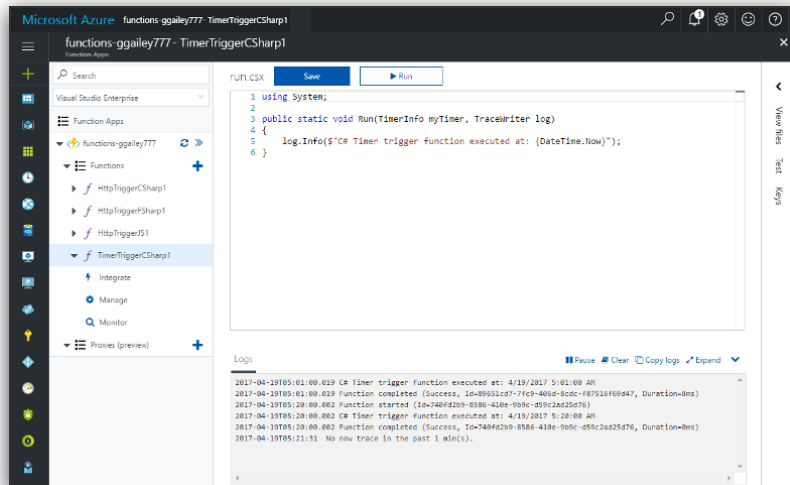
Keep the development loop fast

Developing and testing within the production-level cloud platform can be slow.



Limitations of online serverless development

- Most cloud providers offer online editors for functions.
- Great for getting started.
- Limitations:
 - Limited functionality
 - Missing typical IDE functionality
 - Minimal version control



Develop locally

Many (third-party) projects aim to improve the local development experience:

- AWS SAM—<https://aws.amazon.com/serverless/sam/>
- Serverless Framework—<https://github.com/serverless/serverless>
- Azure Core Tools—<https://github.com/Azure/azure-functions-core-tools>

Test locally (1): FaaS emulators

Speed up testing by deploying a local FaaS emulator.

Emulators exist for most major FaaS providers:

- Azure Function Core Tools—<https://github.com/Azure/azure-functions-core-tools>
- Google Cloud Function Emulator—<https://cloud.google.com/functions/docs/emulator>
- serverless.com emulator—<https://github.com/serverless/emulator>
- AWS SAM local—<https://github.com/awslabs/aws-sam-cli>
- Docker Lambda—<https://github.com/lambci/docker-lambda>

Test locally (2): cloud emulators

Attempts to emulate FaaS and related services:

- localstack—<https://github.com/localstack/localstack>
- AWS SAM local—<https://github.com/awslabs/aws-sam-cli>

⚠ Can become a time sink to get and keep working.

Test locally (3): open-source FaaS platforms

Open-source FaaS platforms allow you to fully replicate the platform locally.

CNCF Serverless Landscape
2019-05-19T02:14:19Z fe939f1

See the serverless interactive display at s.cncf.io

Greyed logos are not open source

Tools

- CloudBeaver, dashbird, Epsagon, event gateway, HADURA, IO|pipe, Iron.io, Node Lambda, python-λ, SCAR, SIGMA, STACKERY, THUNDER

Security

- Intrinsic, Protego, PURESEC, TRIDENT STACK

Framework

- ΔPEX, .arc Architect, aws Chalice, AWS SAM, Cloudify, FLOJO, gas.io, serverless, SPARTA, Spring Cloud Function

Hosted

- IBM Cloud Functions, kit@une, netlify, PaaS@PaaS, spotinst, stdlib, Tencent Cloud Functions, ZEIT

Installable

- OpenFaaS, AppScale, DISPATCH, fission, fn, GALACTIC FOG, Knative, Kubeless, Kyma, LunchBuddy, nuclio, OPERFaaS, PipelineAI, PaaS@PaaS, Virtual Kubelet

Cloud Native Landscape

Serverless computing refers to a new model of cloud native computing, enabled by architectures that do not require server management to build and run applications. This landscape illustrates a finer-grained deployment model where applications, bundled as one or more functions, are uploaded to a platform and then executed, scaled, and billed in response to the exact demand needed at the moment

s.cncf.io

CLOUD NATIVE Landscape
CLOUD NATIVE COMPUTING FOUNDATION
Redpoint

source: <https://github.com/cncf/wg-serverless>

Live-reloading

Automate function updates to provide instant “frontend”-like feedback.

The image illustrates a live-reloading workflow. On the left, a terminal window shows two processes running. The top window, titled '2. vim', displays a markdown file being edited with the following content:

```
1 ---
2 title: "Going FaaSter: Cost-Performance Optimizations of Serverless on Kubernetes @ Serverless Architecture Conference 2019"
3 layout: post
4 tags:
5 - serverless
6 ---
7
8 FOOBAR
9
```

The bottom window, titled '1. fsevent_watch', shows the configuration and status of the live-reloading process:

```
Configuration file: /Users/erwin/Github/website/_config.yml
Source: /Users/erwin/Github/website
Destination: /Users/erwin/Github/website/_site
Incremental build: disabled. Enable with --incremental
Generating...
done in 0.957 seconds.
Auto-regeneration: enabled for '/Users/erwin/Github/website'
Configuration file: /Users/erwin/Github/website/_config.yml
Server address: http://127.0.0.1:4000/
Server running... press ctrl-c to stop.
Regenerating: 1 file(s) changed
```

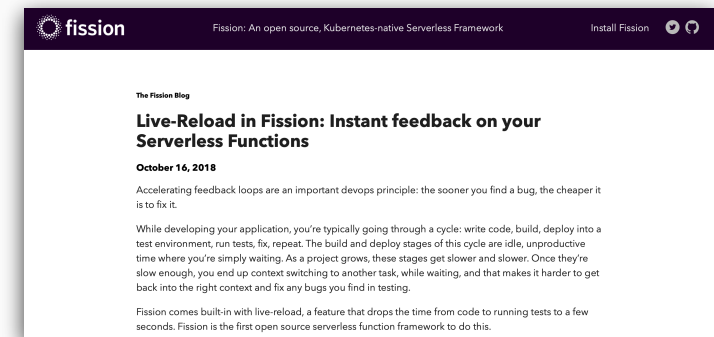
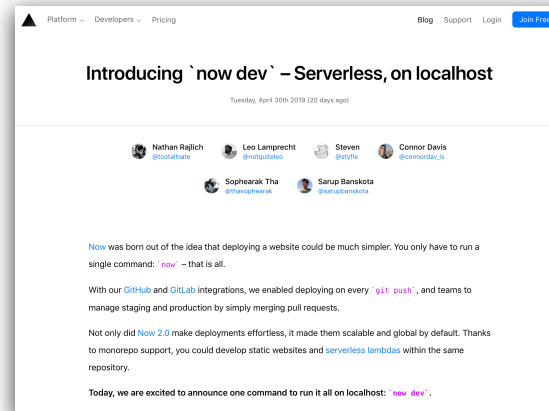
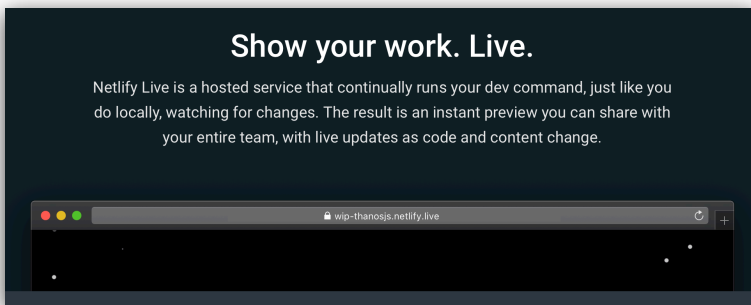
On the right, a browser window shows the rendered output at the address `127.0.0.1:4000/posts/`. The page displays a list of posts under the heading 'DEFERRED POSTS'. A 'menu' button is visible. The 'All Posts' section shows a list of posts for the year '2019', including the post 'Going FaaSter: Cost-Performance Optimizations of Serverless on Kubernetes @ Serverless Architecture Conference 2019' with the content 'FOOBAR', and 'Four Techniques Serverless Platforms Use to Balance Performance'.

Live-reloading in FaaS platforms

DIY basic live-reloading:

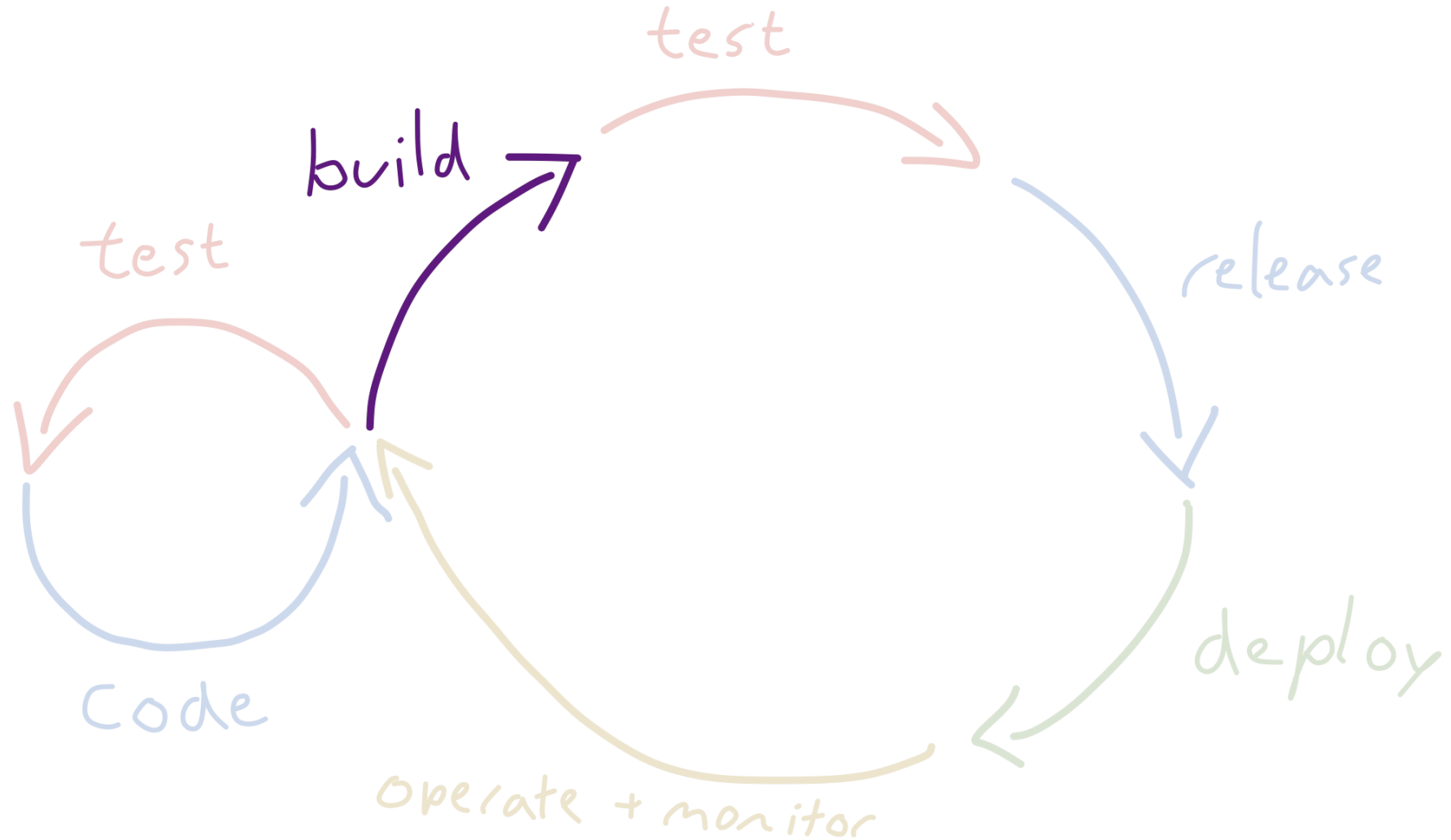
```
fswatch /my/serverless/project | deploy_dev_function.sh
```

Or, built-in:

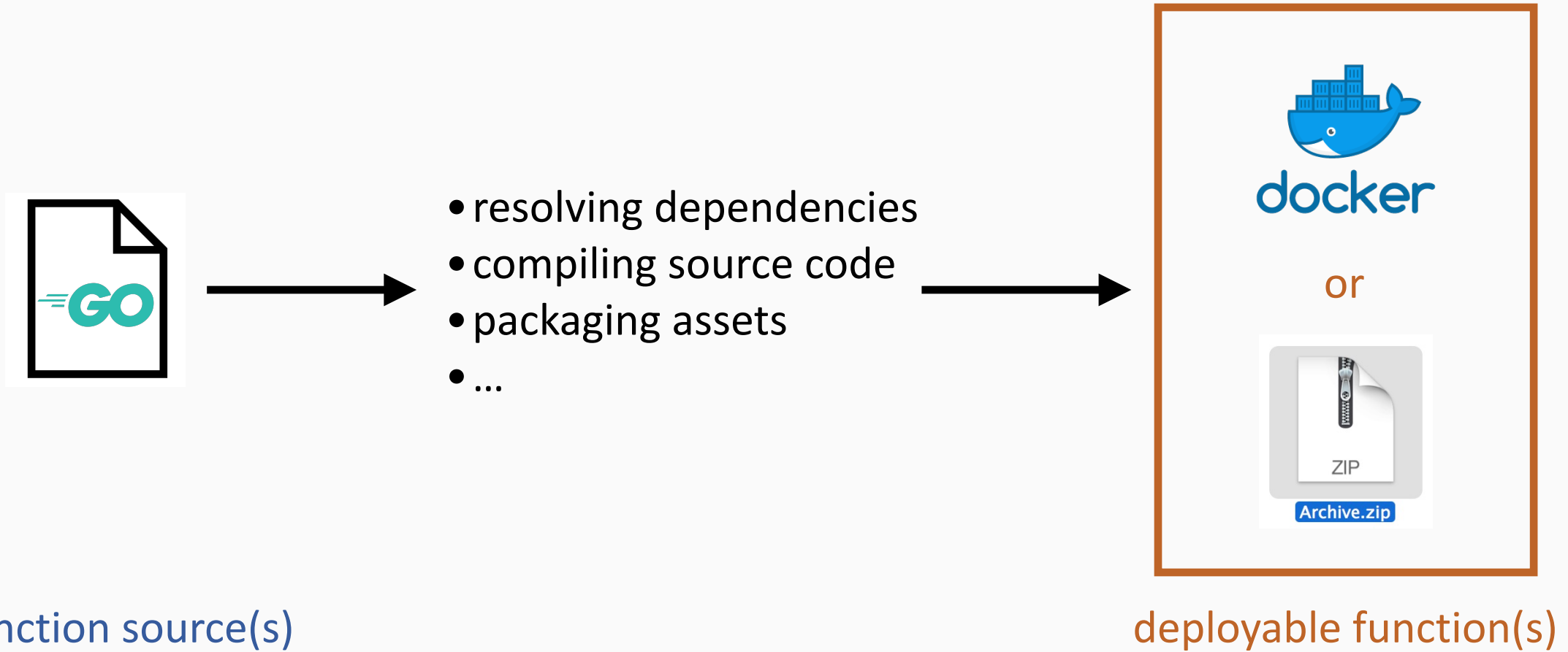


Operation Loop

build.

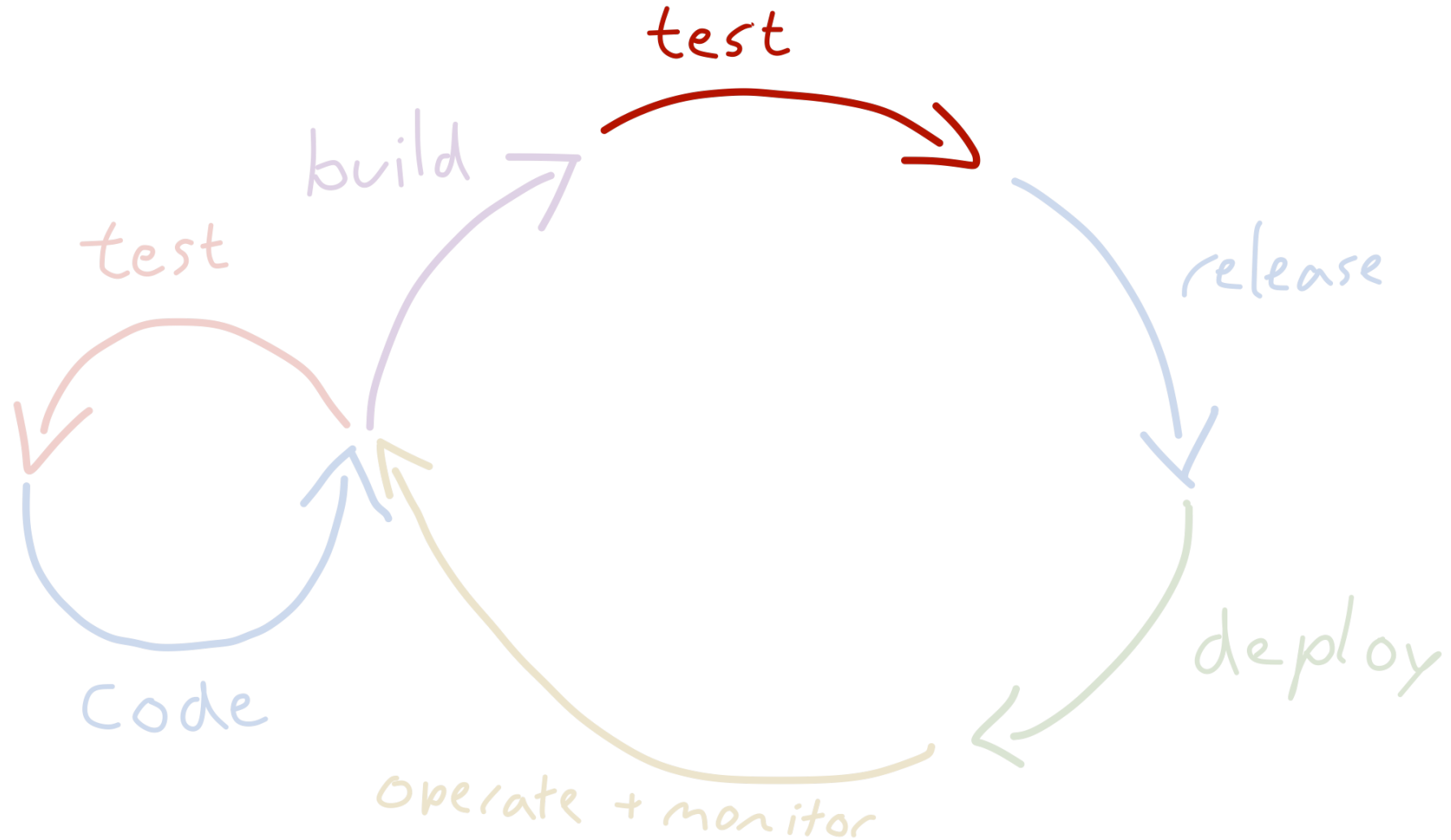


Automated, reproducible builds

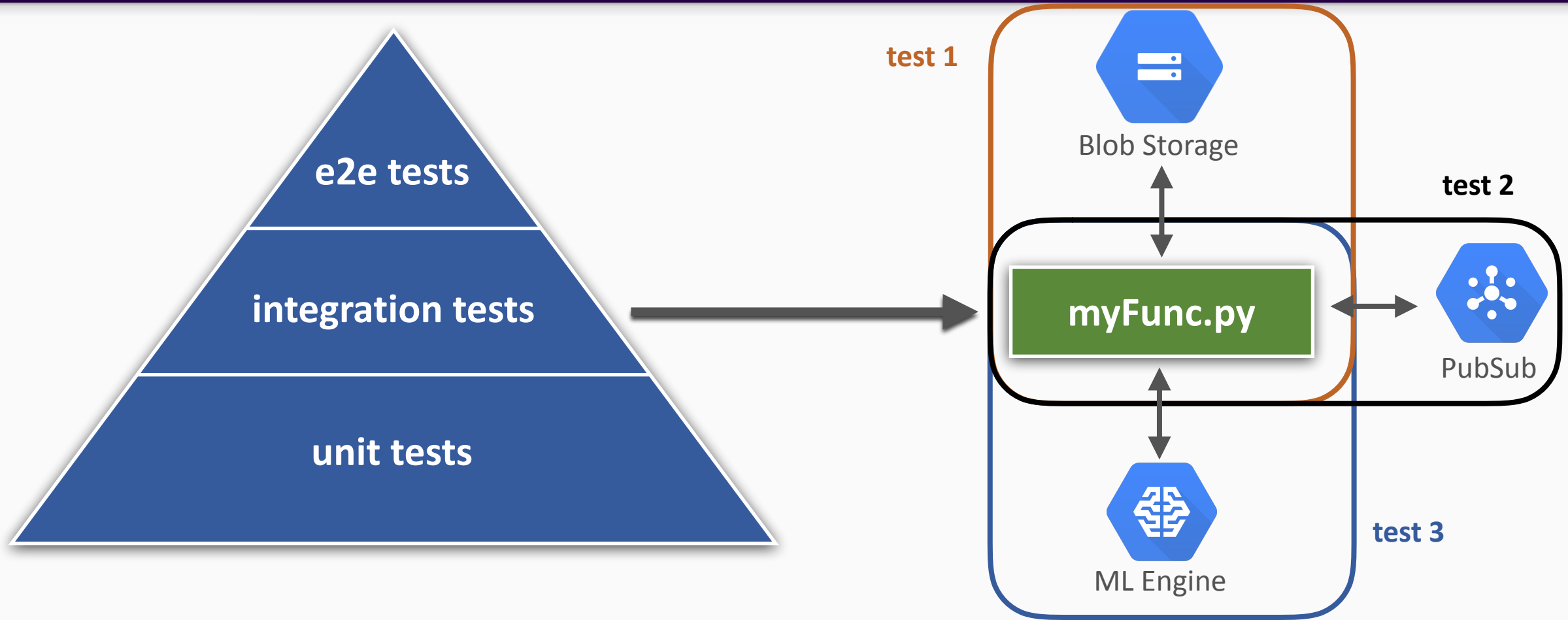


Operation Loop

test.



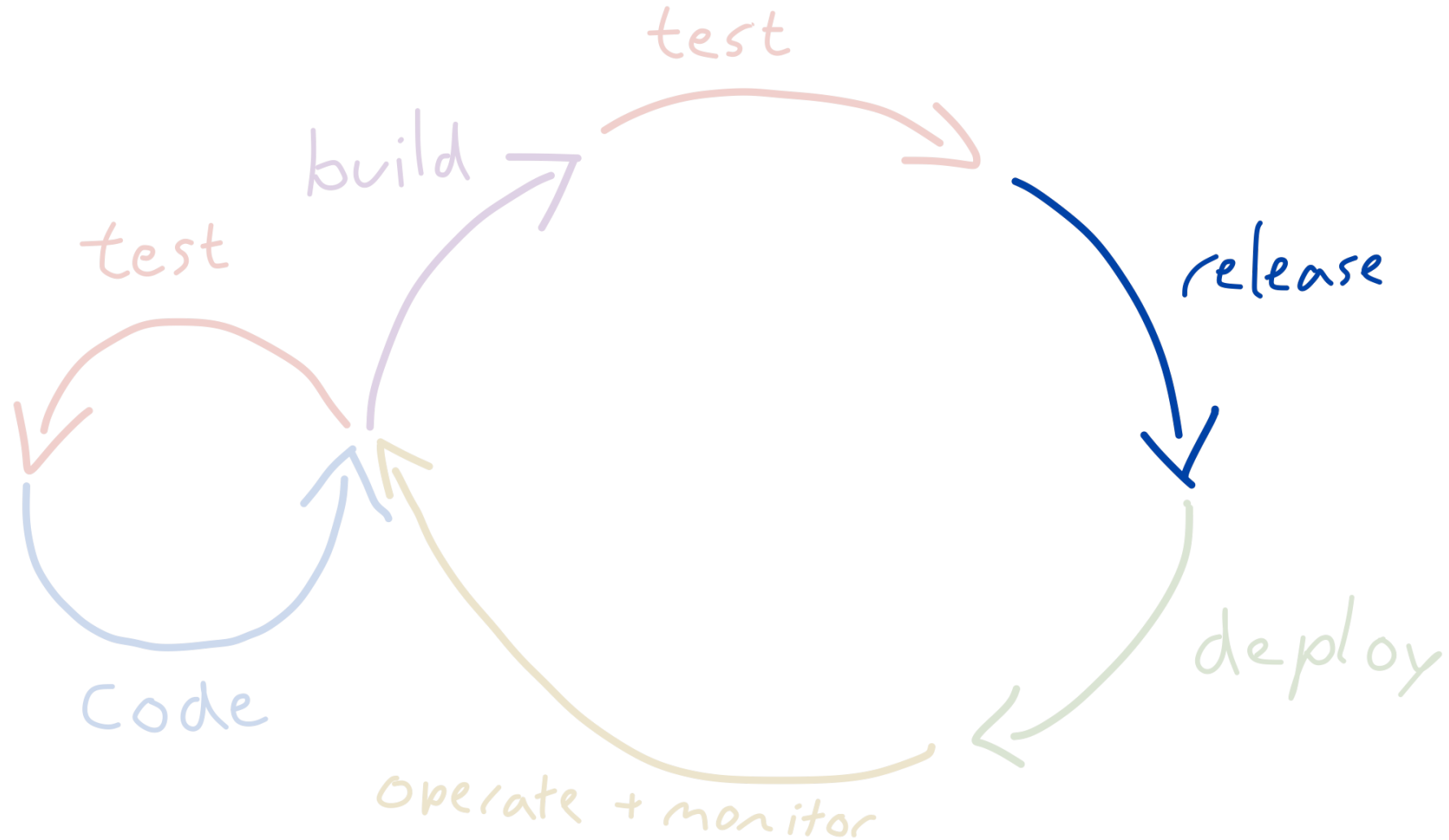
Comprehensive testing in a serverless world



- Avoid hard-coding service-related details.
- Fully isolate the testing from production.
- Set limits and alerts (for run-away functions!).

Operation Loop

release.



“Operations by pull request”



Imperative configuration...

deploy.sh

```
# [edit counter.py]
fission environment create --name python --image fission/python-env
fission function create --name counter --env python --src ./counter.py
fission trigger create --name counter-get --method GET --url /counter
```

...vs. declarative configuration

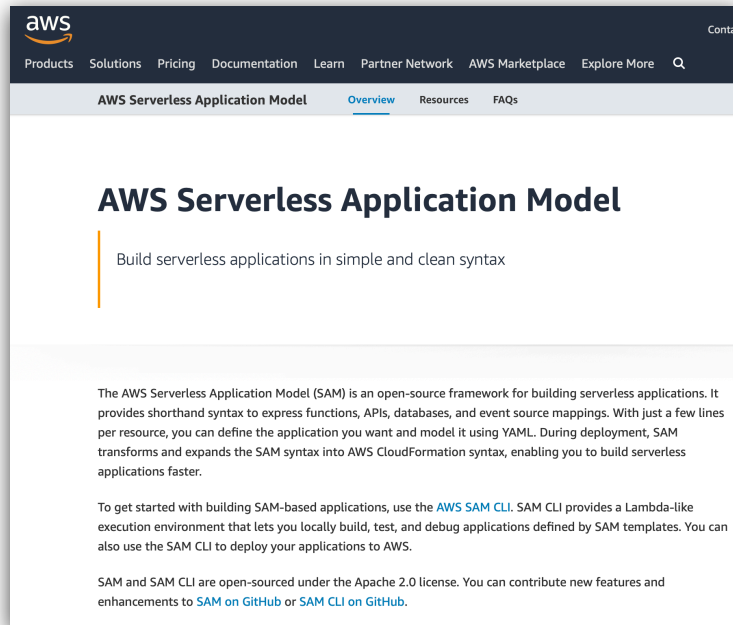
```
apiVersion: fission.io/v1
kind: Environment
metadata:
  name: python
  namespace: default
spec:
  version: 2
  runtime:
    image: fission/python-env:1.1.0
---
apiVersion: fission.io/v1
kind: Function
metadata:
  name: counter
  namespace: default
spec:
  environment:
    name: python
    namespace: default
# [...]
```

deploy.sh

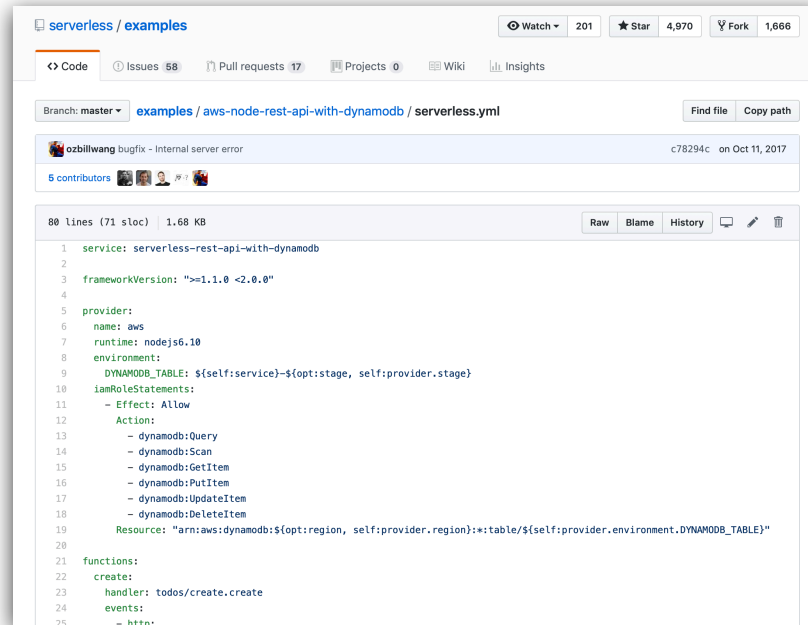
```
# [edit counter.py and counter.yaml]
# create function deployment specs:
kubectl apply -f counter.yaml

# [edit counter.py]
# OR, using the fission API:
fission spec init
fission spec apply # Generates and applies Kubernetes specs
```

Examples of projects focusing on declarative configurations



AWS SAM



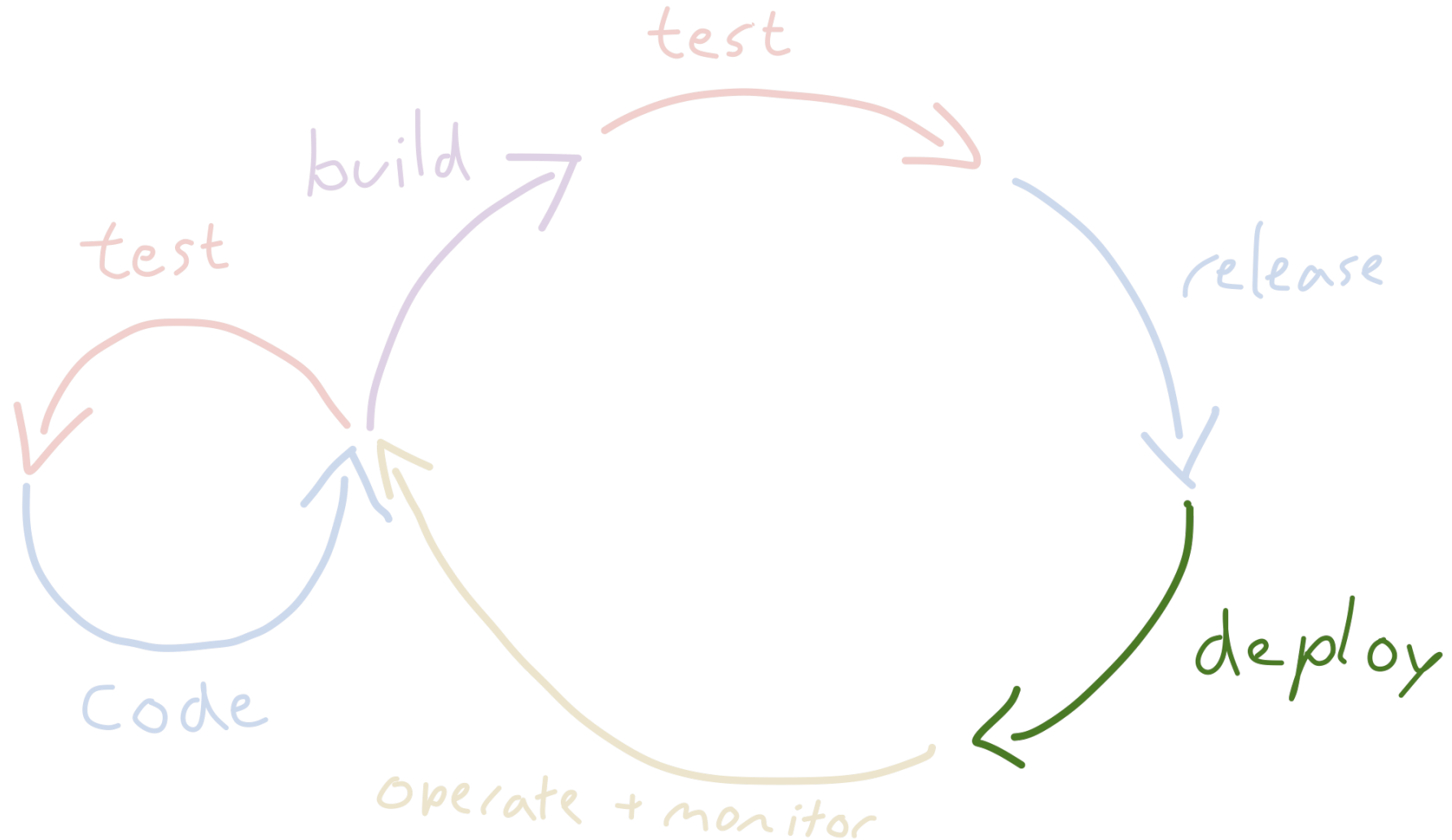
Serverless framework



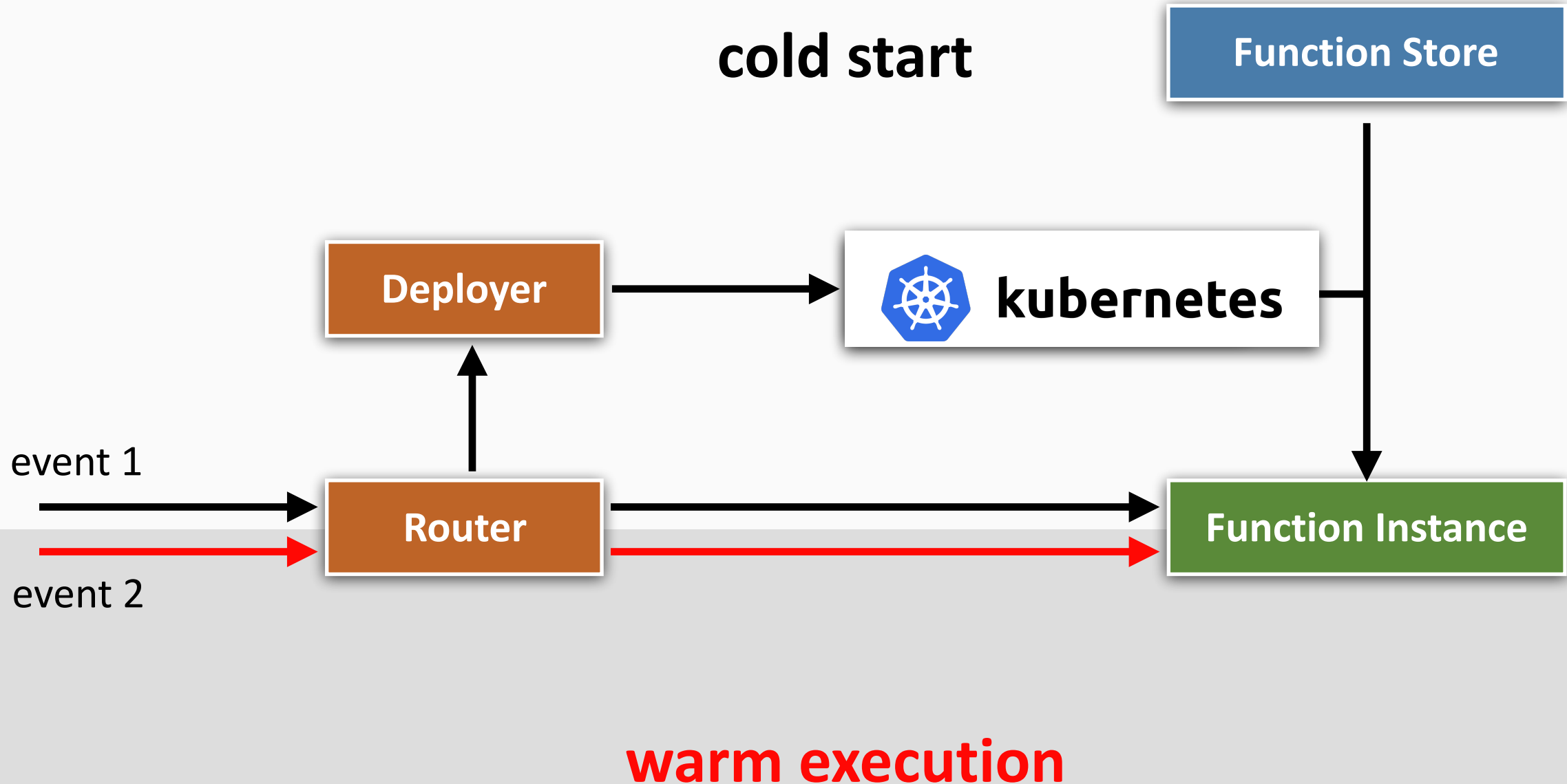
Most open-source FaaS platforms

Operation Loop

deploy.



Understanding the FaaS deployment process

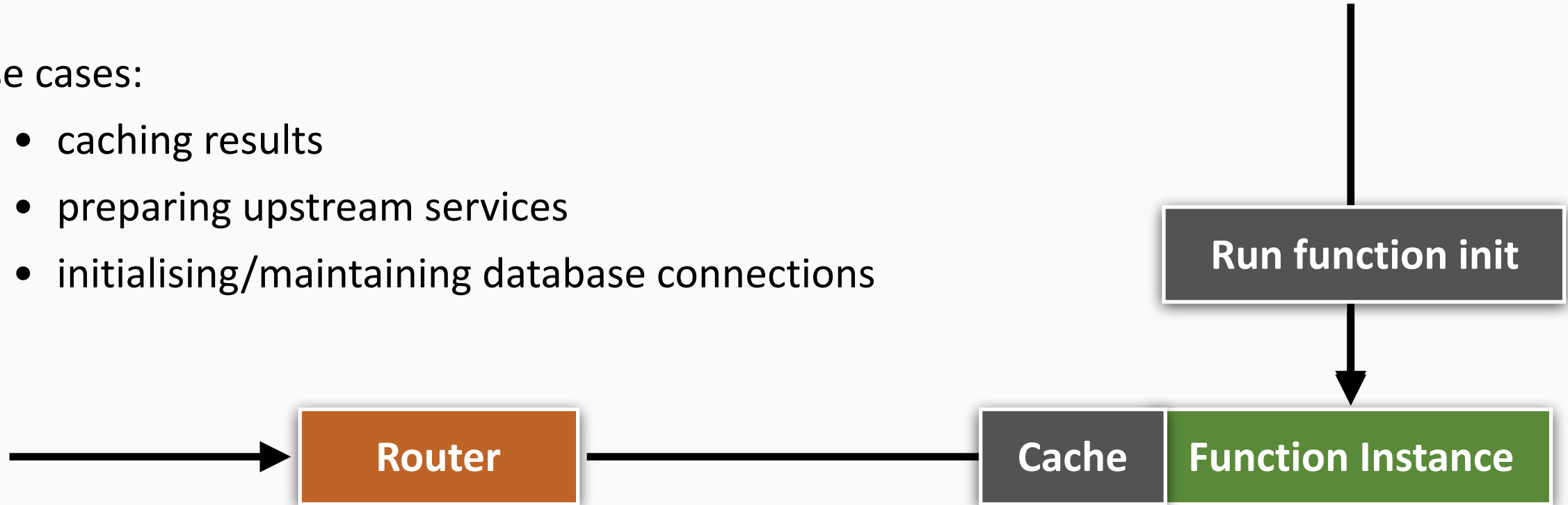


Caching in deployed function instances

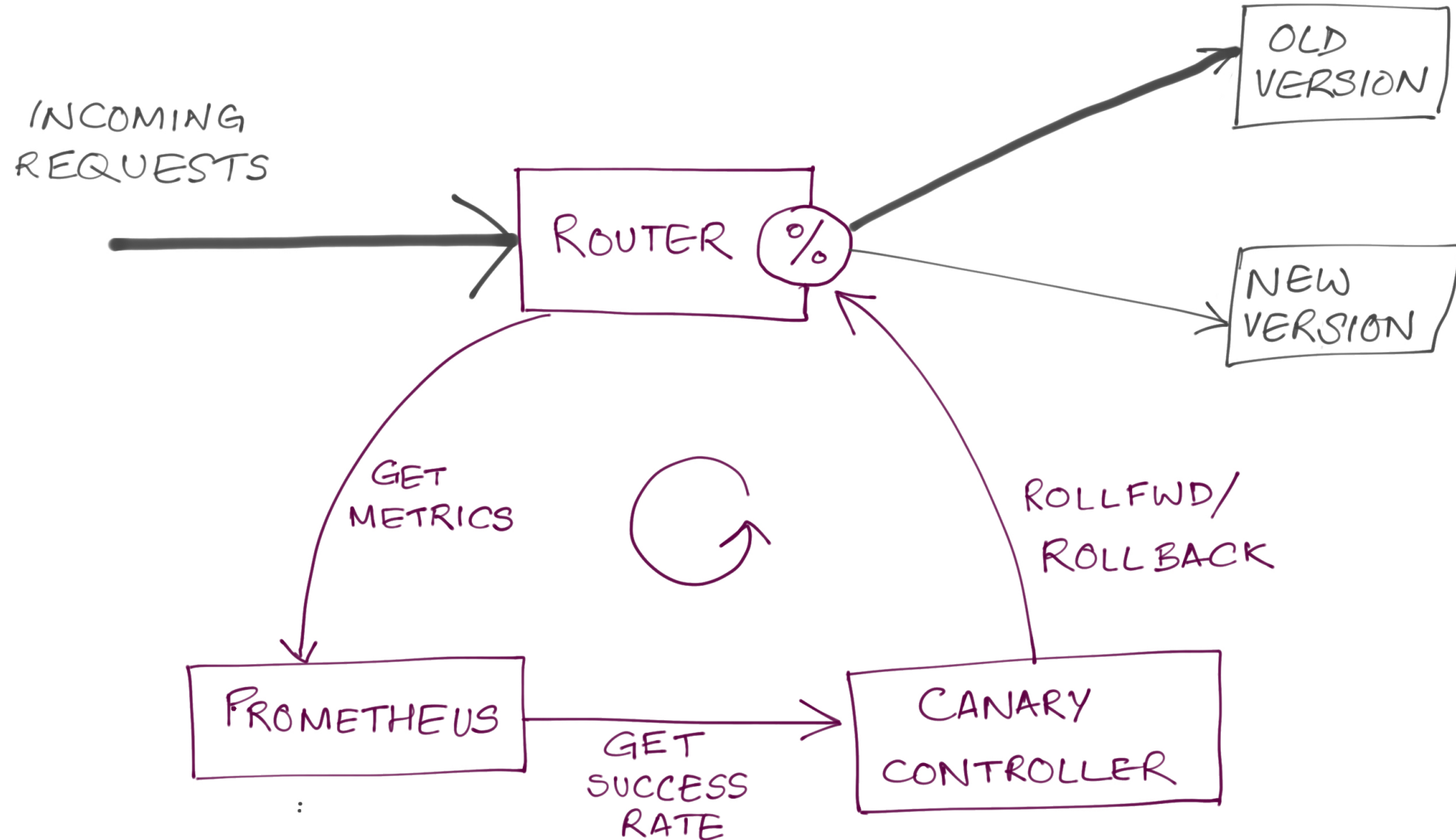
Functions are stateless, but can maintain non-persistent state.

Use cases:

- caching results
- preparing upstream services
- initialising/maintaining database connections



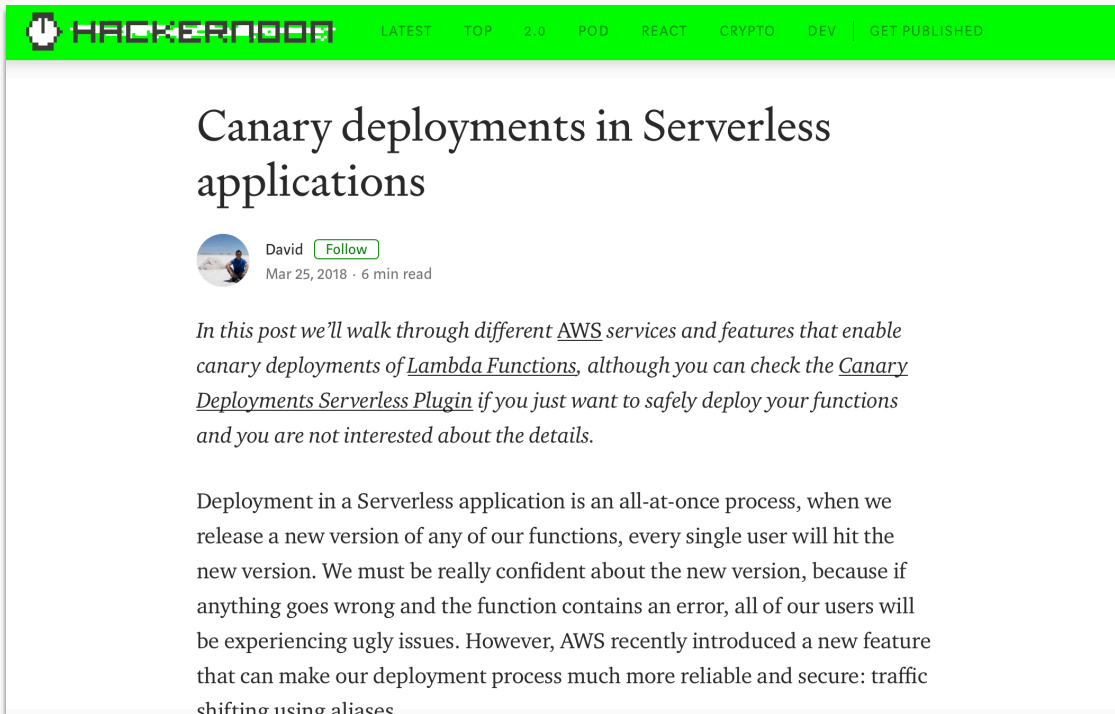
Canary Deployments



Canary Deployments (2)



Canary Deployments (3)



HACKERNOON LATEST TOP 2.0 POD REACT CRYPTO DEV | GET PUBLISHED

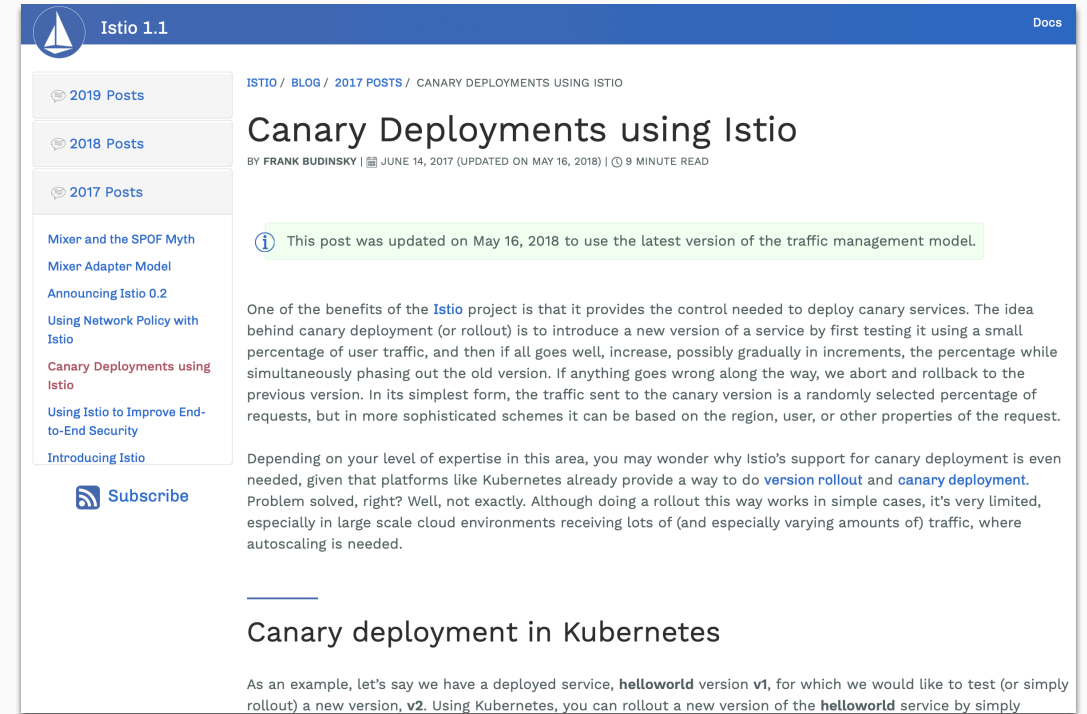
Canary deployments in Serverless applications

David [Follow](#)
Mar 25, 2018 · 6 min read

In this post we'll walk through different [AWS](#) services and features that enable canary deployments of [Lambda Functions](#), although you can check the [Canary Deployments Serverless Plugin](#) if you just want to safely deploy your functions and you are not interested about the details.

Deployment in a Serverless application is an all-at-once process, when we release a new version of any of our functions, every single user will hit the new version. We must be really confident about the new version, because if anything goes wrong and the function contains an error, all of our users will be experiencing ugly issues. However, AWS recently introduced a new feature that can make our deployment process much more reliable and secure: traffic shifting using aliases

AWS CodeDeploy + AWS Lambda



Istio 1.1 Docs

2019 Posts
2018 Posts
2017 Posts

[Mixer and the SPOF Myth](#)
[Mixer Adapter Model](#)
[Announcing Istio 0.2](#)
[Using Network Policy with Istio](#)
[Canary Deployments using Istio](#)
[Using Istio to Improve End-to-End Security](#)
[Introducing Istio](#)

[Subscribe](#)

Canary Deployments using Istio

BY FRANK BUDINSKY | JUNE 14, 2017 (UPDATED ON MAY 16, 2018) | 9 MINUTE READ

This post was updated on May 16, 2018 to use the latest version of the traffic management model.

One of the benefits of the [Istio](#) project is that it provides the control needed to deploy canary services. The idea behind canary deployment (or rollout) is to introduce a new version of a service by first testing it using a small percentage of user traffic, and then if all goes well, increase, possibly gradually in increments, the percentage while simultaneously phasing out the old version. If anything goes wrong along the way, we abort and rollback to the previous version. In its simplest form, the traffic sent to the canary version is a randomly selected percentage of requests, but in more sophisticated schemes it can be based on the region, user, or other properties of the request.

Depending on your level of expertise in this area, you may wonder why Istio's support for canary deployment is even needed, given that platforms like Kubernetes already provide a way to do [version rollout](#) and [canary deployment](#). Problem solved, right? Well, not exactly. Although doing a rollout this way works in simple cases, it's very limited, especially in large scale cloud environments receiving lots of (and especially varying amounts of) traffic, where autoscaling is needed.

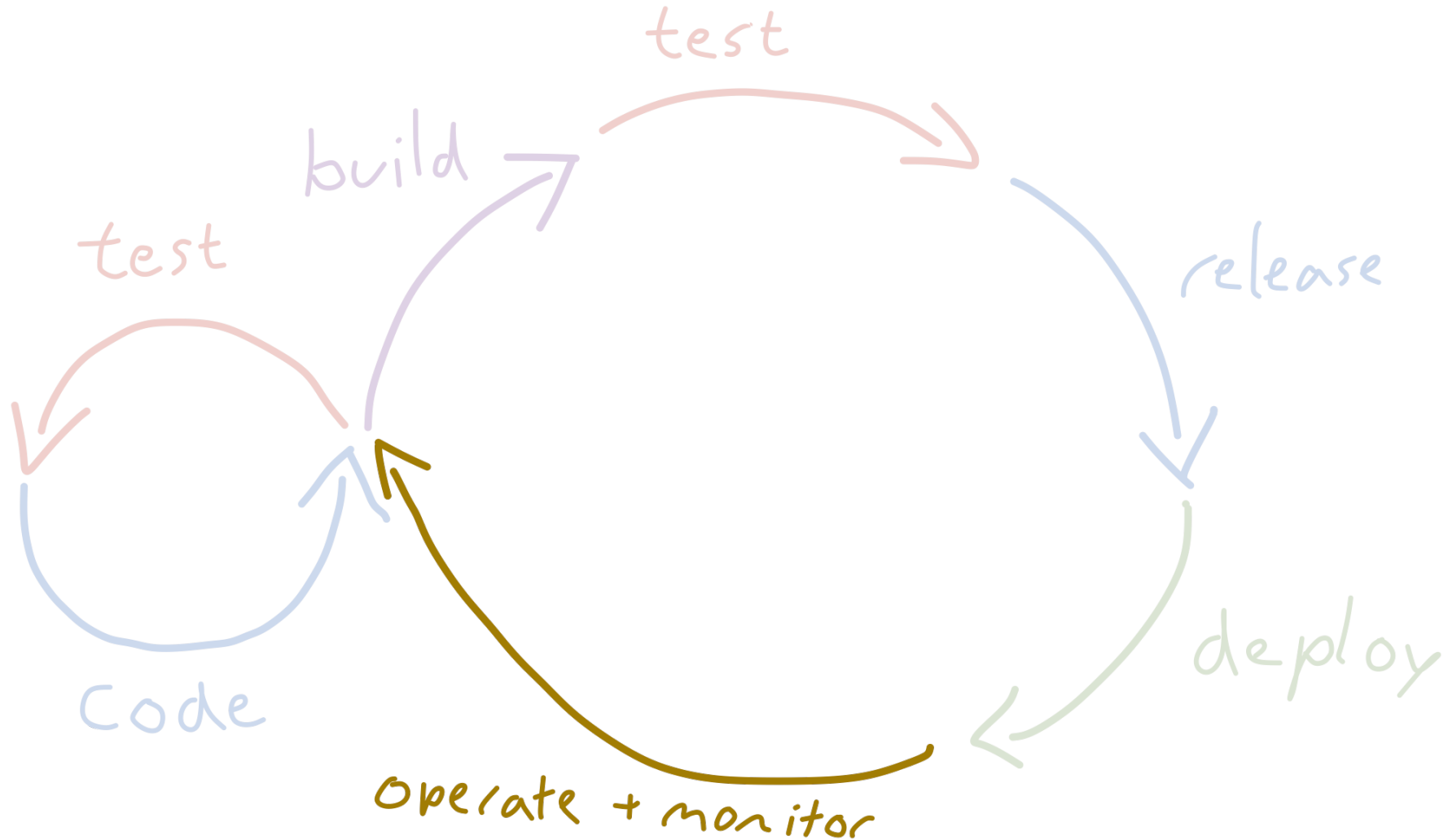
Canary deployment in Kubernetes

As an example, let's say we have a deployed service, **helloworld** version **v1**, for which we would like to test (or simply rollout) a new version, **v2**. Using Kubernetes, you can rollout a new version of the **helloworld** service by simply

Service mesh-based FaaS platforms (e.g., knative)

Operation Loop

operate, monitor.



Monitoring, tracing, logging

Serverless applications still need to be monitored

Serverless platforms help with...

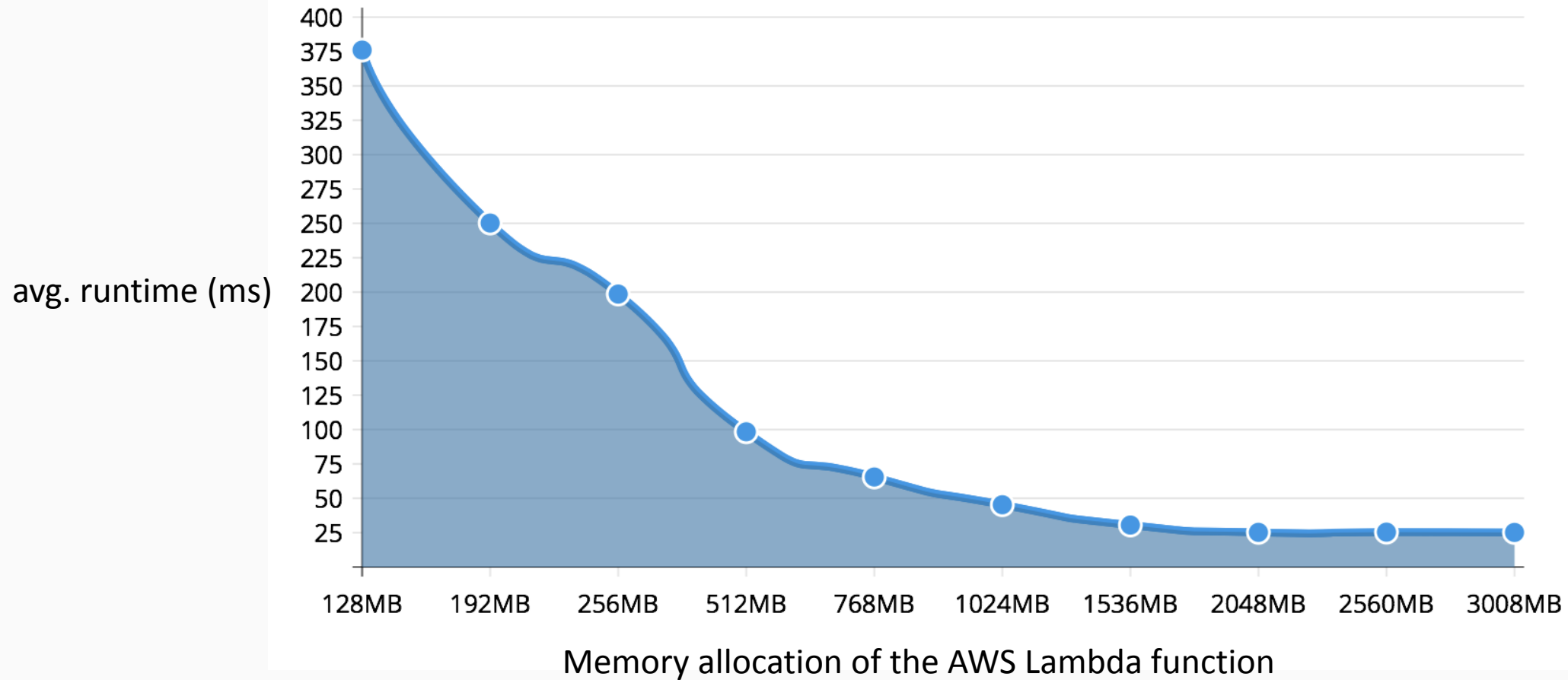
- metric monitoring (system-level and some user-level metrics)
- log aggregation
- distributed tracing

Potential pitfalls:

- (implicit) costs
- proprietary formats
- vendor lock-in

Memory allocation is linked to CPU/IO shares

Average runtime calculating Fibonacci numbers 1 to 100



Never scaling down to zero

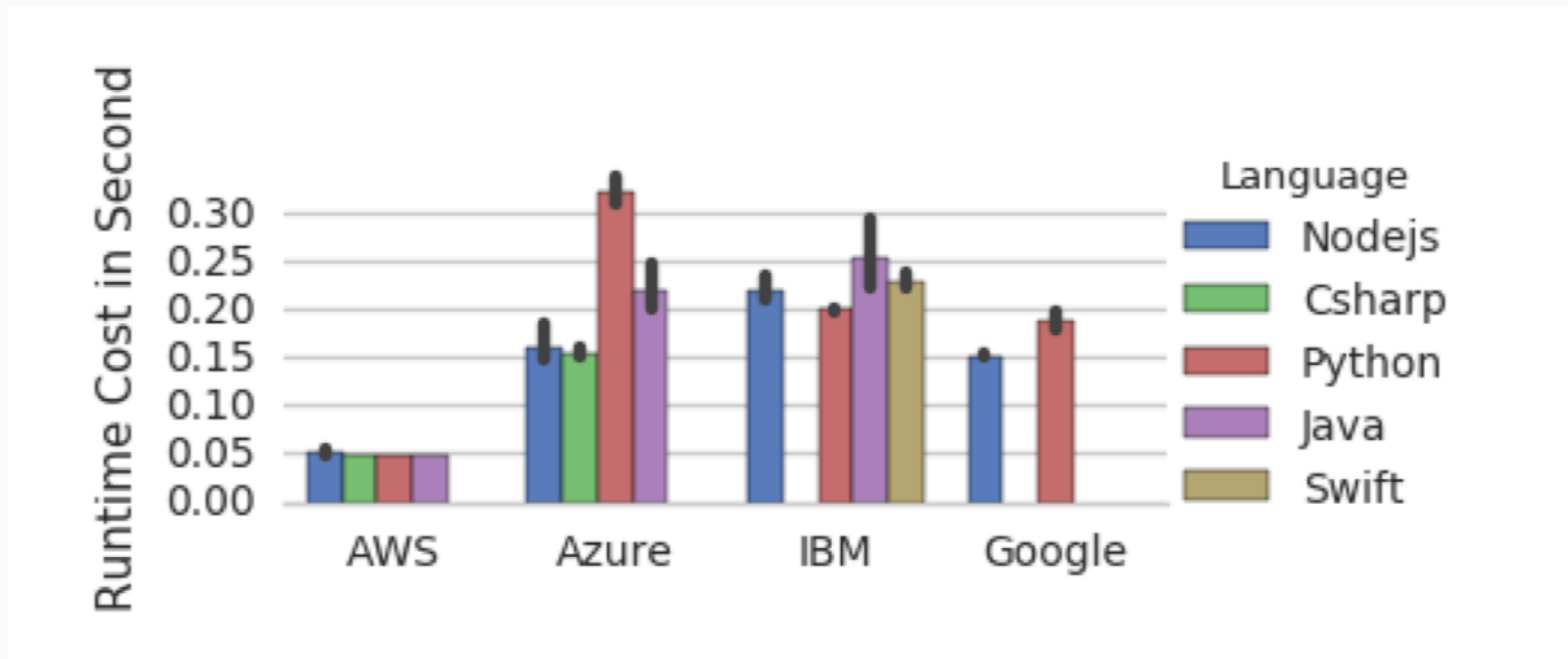
Keep function instances alive at all times to avoid (most) cold starts.

- Not well supported in managed FaaS platforms.
 - Azure supports minimal instances in its “premium plan”.
- Well supported in open-source FaaS platforms.



The language can impact performance

- In theory, languages should perform similar.
- In practice, maturity of languages differs.



Conclusion

Lessons learned:

1. Keep the serverless development loop as fast as possible.
2. Prefer declarative over imperative configuration.
3. Automate the build and deployment process (canaries!).
4. Use the function deployment process to your advantage.
5. Be wary of performance effects of function configuration changes.

Takeaway: The DevOps lifecycle is (still) relevant in a serverless world!

Thanks!



Slack

Twitter

<http://fission.io> + <https://github.com/fission>

<http://slack.fission.io/>

@fissionio

Erwin van Eyk

Software Engineer @ Platform9 Systems

Chair @ SPEC CLOUD RG Serverless

Researcher @ AtLarge Research

@erwinvaneyk

erwin@platform9.com

<https://erwinvaneyk.nl>



Further reading

- More on **FaaS internals and performance**:

- My talk at KubeCon China 2018: <https://erwinvaneyk.nl/kubecon-china-2018-serverless-performance/>
- Or, the blogpost derived from that talk: <https://www.infoq.com/articles/serverless-performance-cost>
- Wang, Liang, et al. "Peeking behind the curtains of serverless platforms." 2018. <https://www.usenix.org/system/files/conference/atc18/atc18-wang-liang.pdf>

- More on **serverless concepts**:

- Serverless is More (2018) - which covers the emergence, current state, and future of serverless: <https://erwinvaneyk.nl/internet-computing-serverless-is-more/>
- CNCF Serverless WG - Serverless Overview Whitepaper and serverless landscape (2017): <https://github.com/cncf/wg-serverless>

Additional Slides

Who is who in serverless computing



Cloud user

- Or just the “**(software) developer**” in the context of this talk.
- Uses cloud (and serverless) services to develop applications.
- **Examples:** Spotify, Netflix, you(?)...



Cloud operator

- Provides cloud (and serverless) services to cloud users.
- Can be a public or private, in-house cloud provider.
- **Examples:** AWS, Platform9, internal DevOps team...



Application user

- Generates events which trigger the execution of (cloud) applications.
- Can be downstream services, or actual (physical) users.
- **Examples:** frontend/UI, workqueue...

Serverless in a nutshell

cloud operator manages...



cloud user (you) manages...

Application

Application

Application

Application

Application
Middleware

Application
Middleware

Application
Middleware

Application
Middleware

Cluster Resource Mgt

Cluster Resource Mgt

Cluster Resource Mgt

Cluster Resource Mgt

Virtualization

Virtualization

Virtualization

Virtualization

Hardware

Hardware

Hardware

Hardware

bare-metal

IaaS



kubernetes

serverless