UseR! 2021
The R Conference

Péter Sólymos

#rinprod

Data science serverless-style with R and OpenFaaS

analythium.io

Peter Solymos
Biologist
Data Scientist

2007: 1st line of R code
2007: 1st R package
2009: 1st *JSS* paper
2010: 1st *R Journal* paper

@psolymos

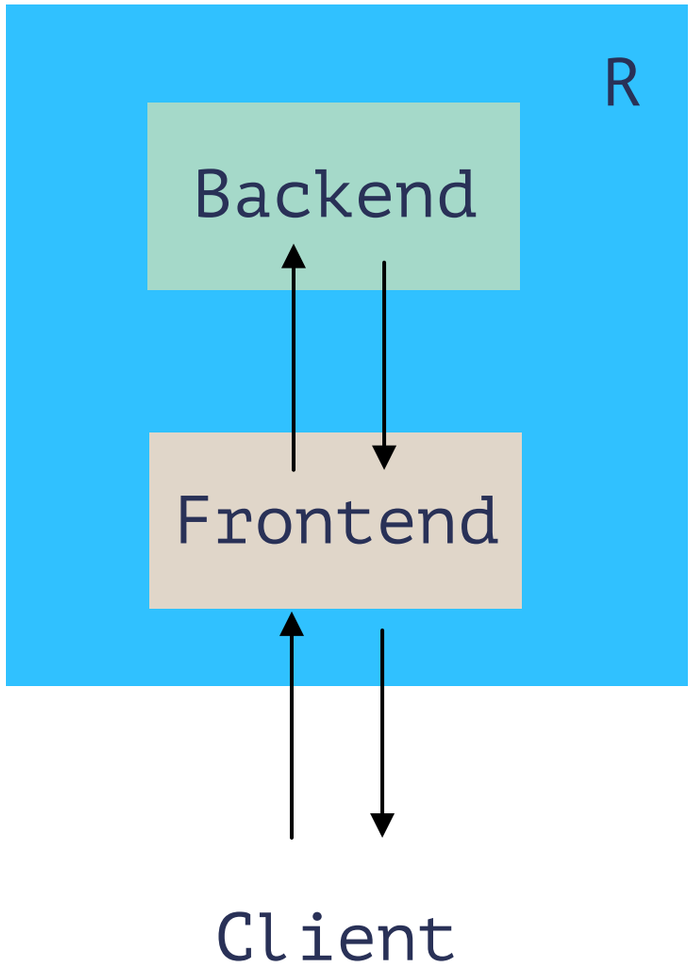
Resilience

Freedom

Simplicity

Why bother?

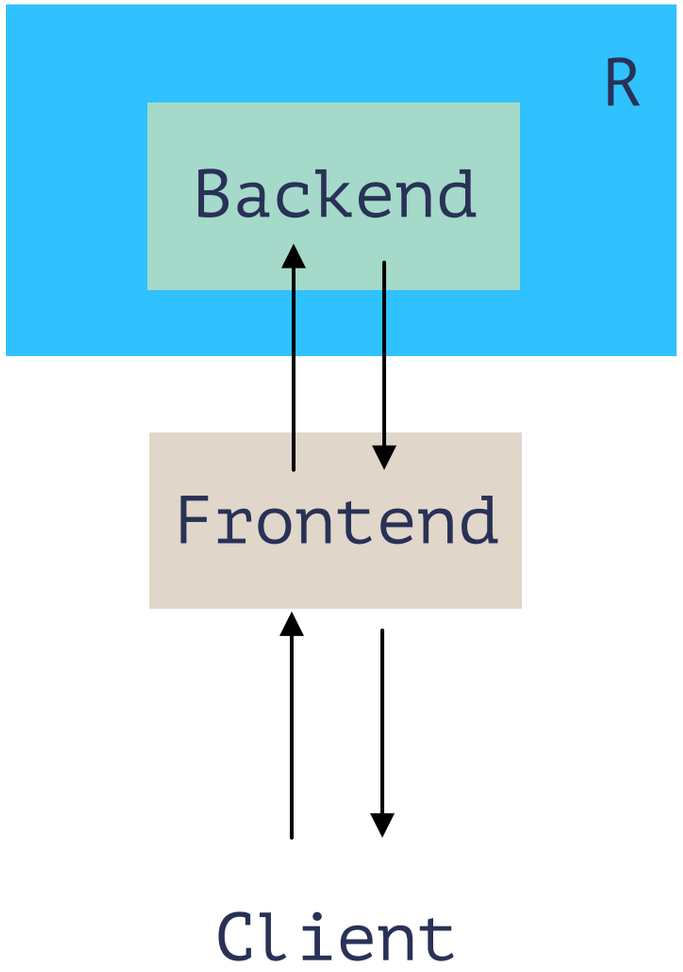
Coupled Analytics



Cost ~ U max(a,b) (B + F)

U: number of users
a: scaling factor for backend
b: scaling factor for frontend
(# instances / user)
B: backend costs
F: frontend costs

Decoupled Analytics



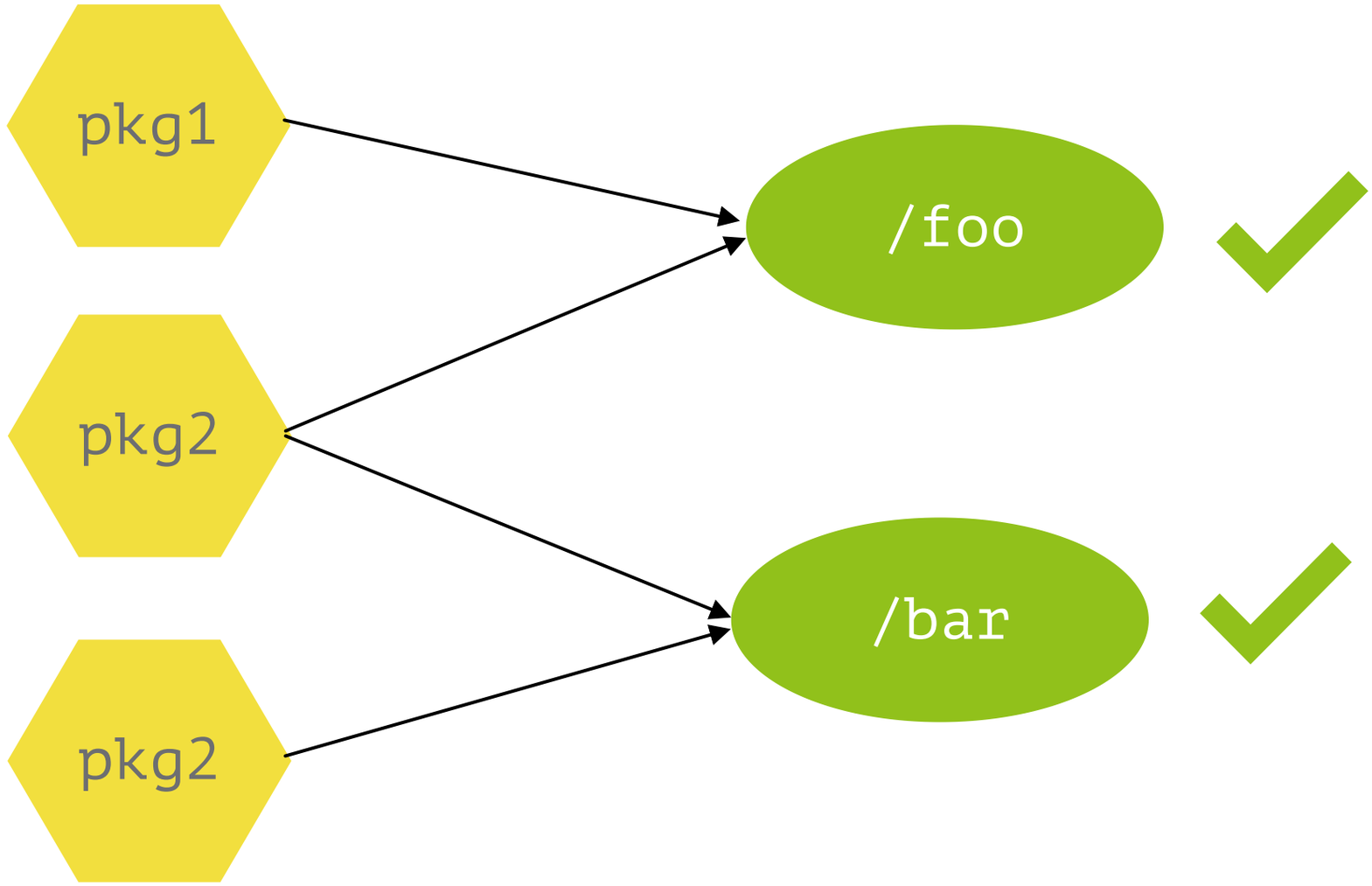
Cost ~ U (aB + bF)

U: number of users
a: scaling factor for backend
b: scaling factor for frontend
(# instances / user)
B: backend costs
F: frontend costs

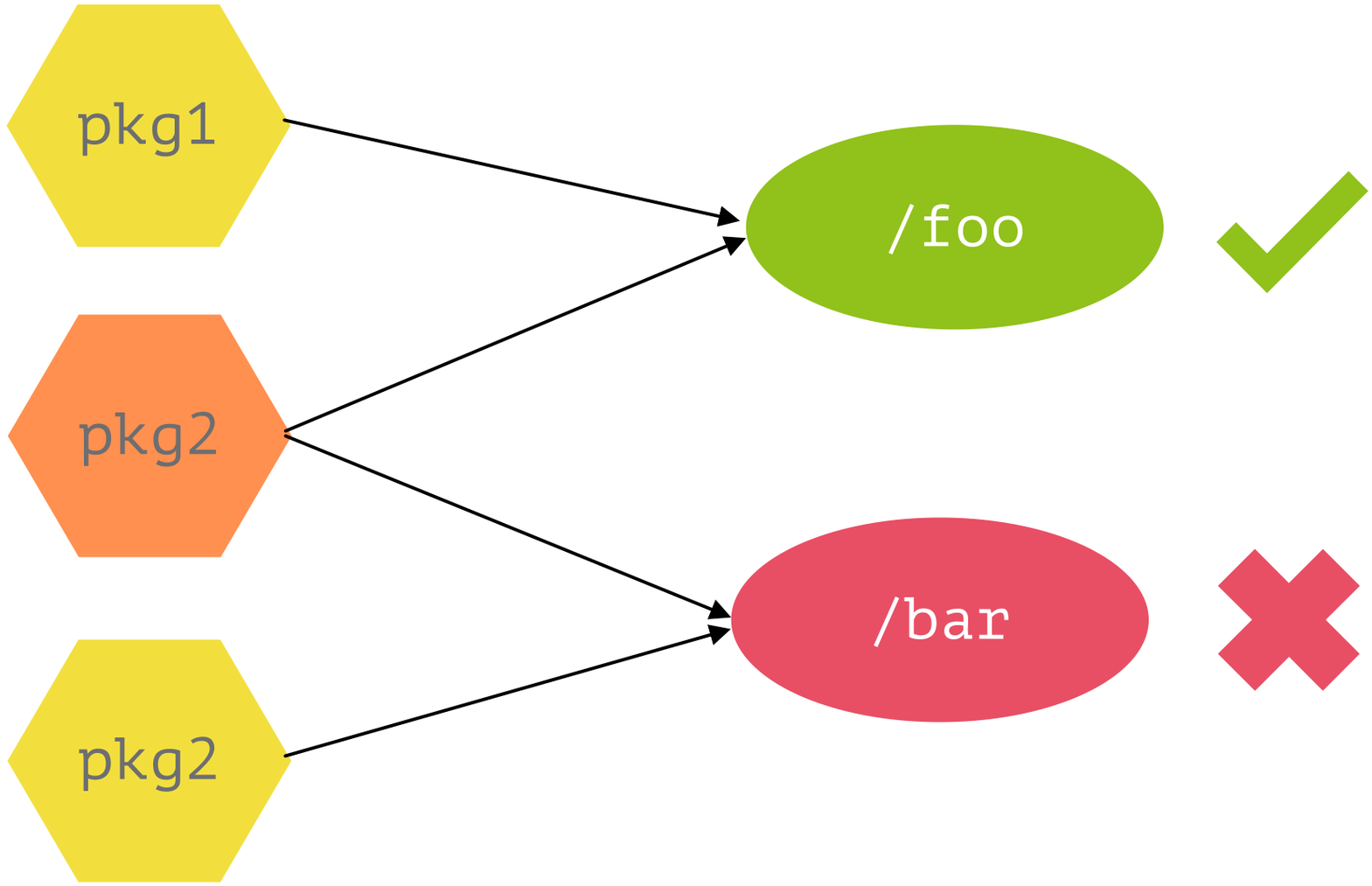
Resilience

Freedom

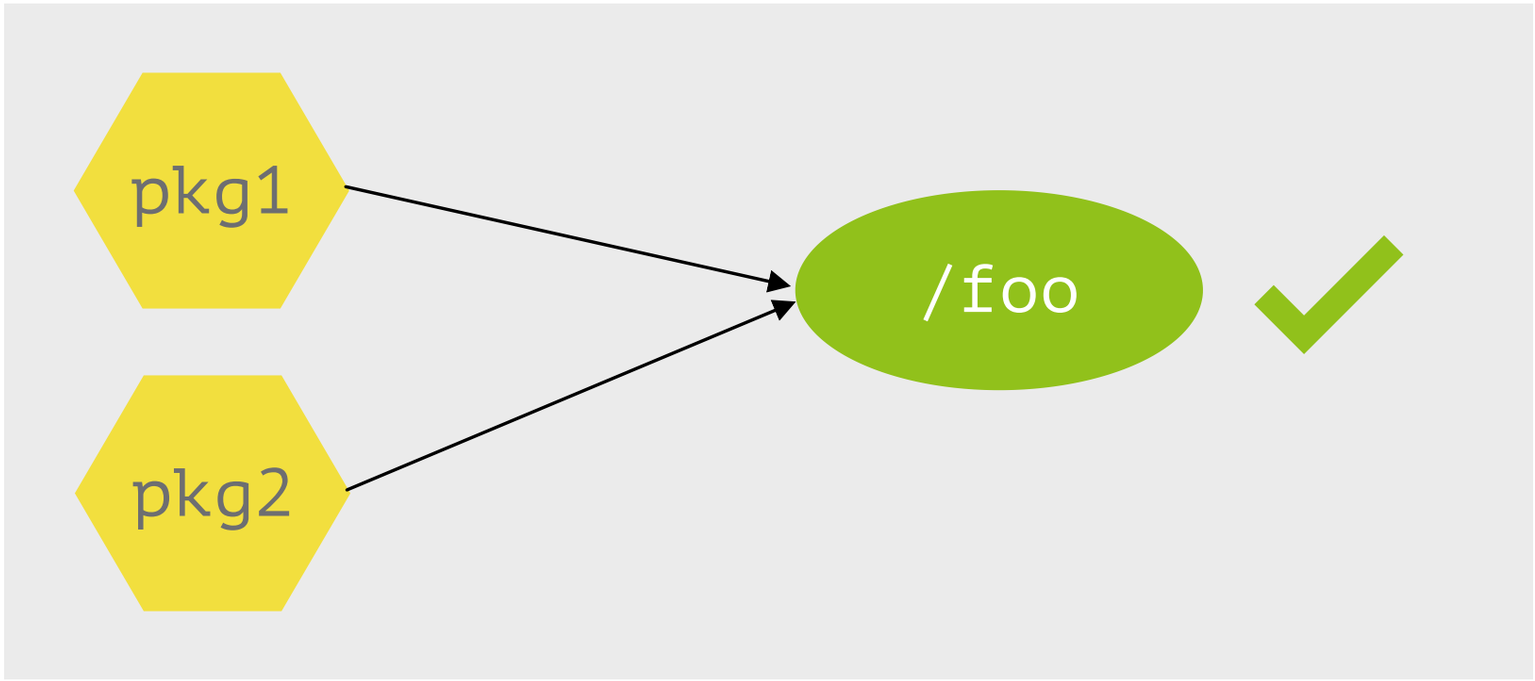
Simplicity

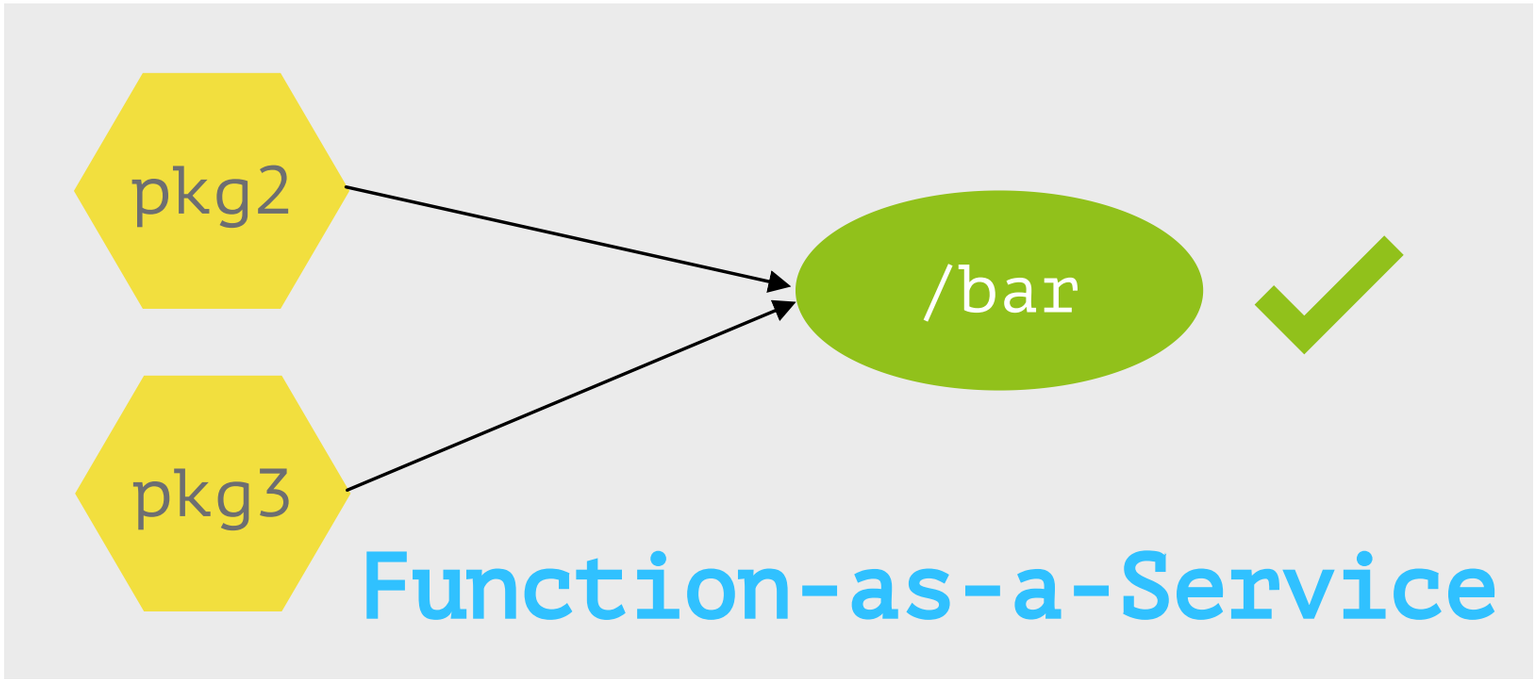


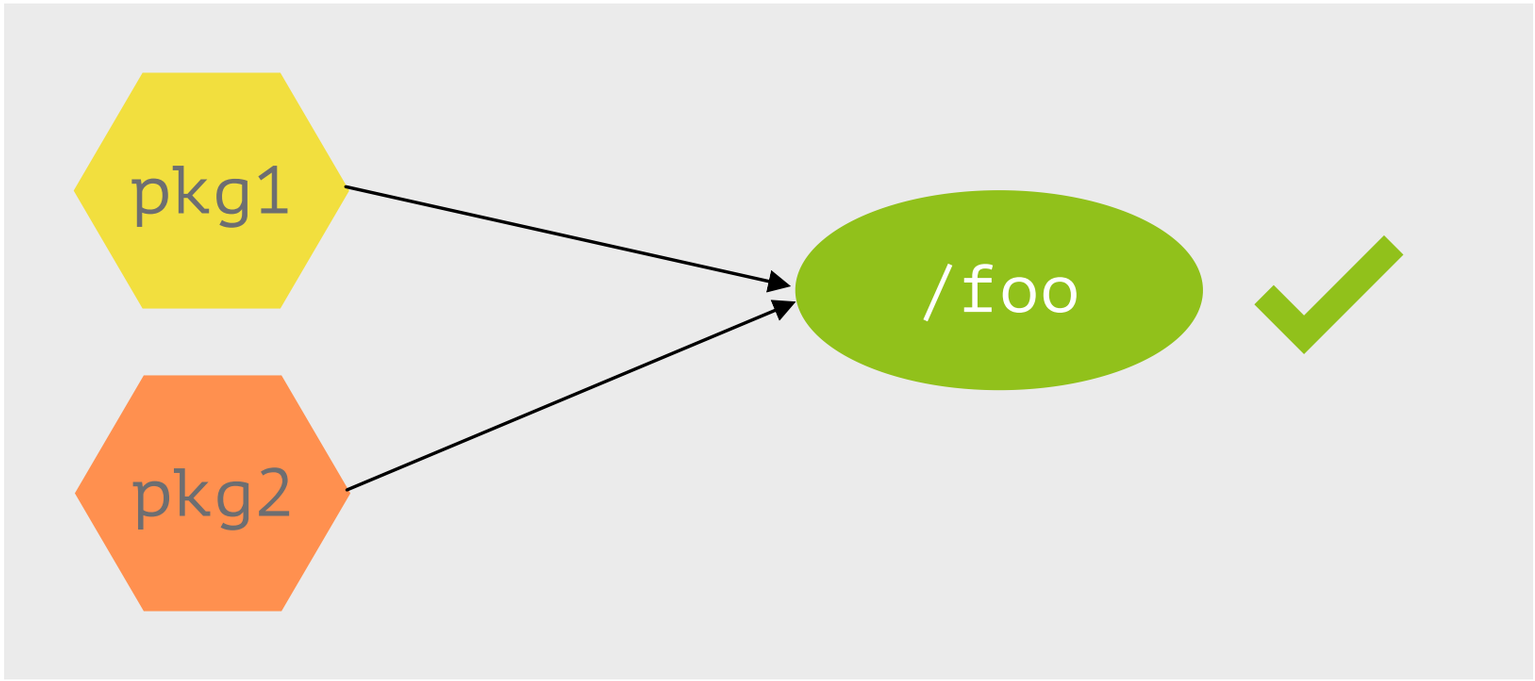
A microservice with mutiple API endpoints

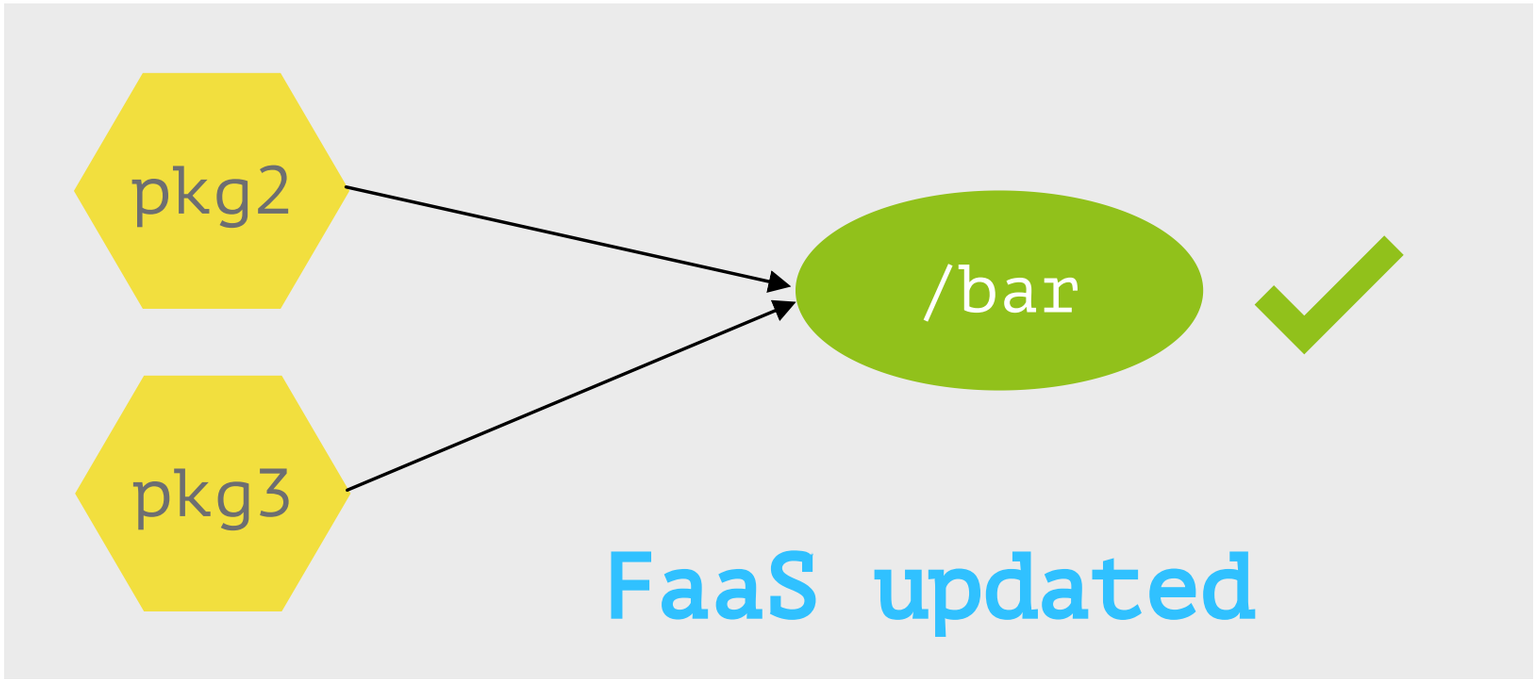


A broken microservice


pkg1
pkg2
/foo


pkg2
pkg3
/bar
Function-as-a-Service


pkg1
pkg2
/foo


pkg2
pkg3
/bar
FaaS updated

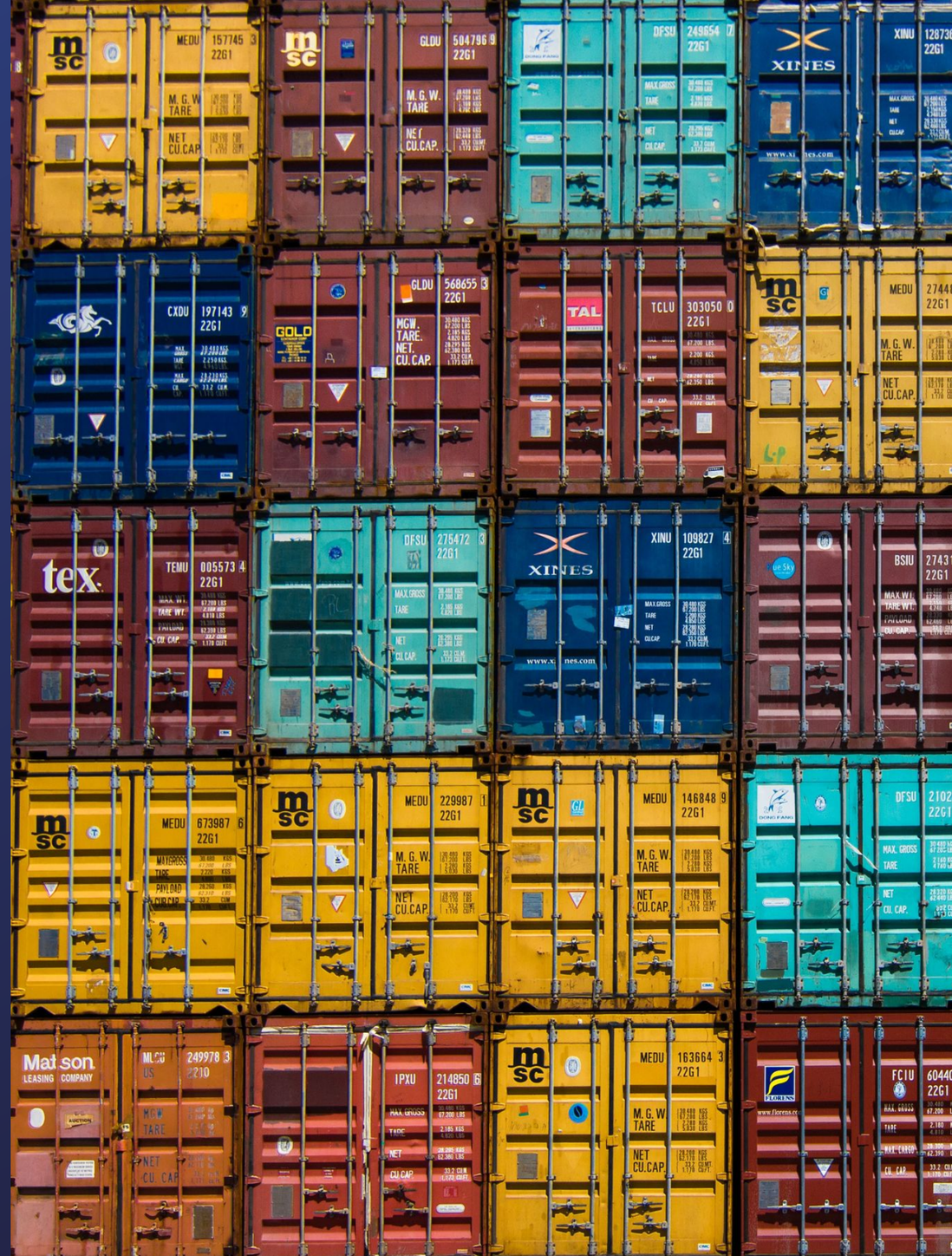
Containers

Immutability

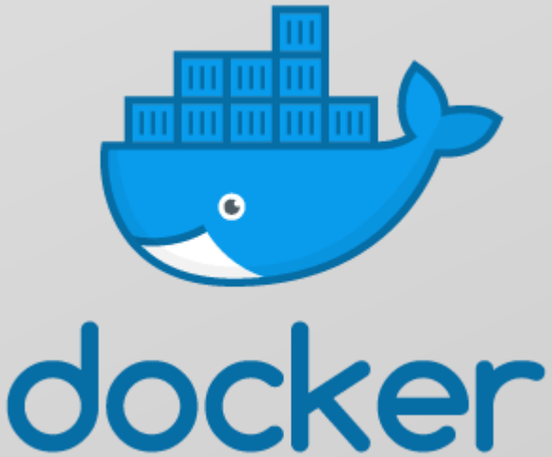
Image layers are immutable
Release, testing, rollback

Isolation

Prevents conflicts
Dedicated resource



DOI 10.32614/RJ-2020-007



The Rockerverse: Packages and Applications for Containerisation with R

by Daniel Nüst, Dirk Eddelbuettel, Dom Bennett, Robrecht Cannoodt, Dav Clark, Gergely Daróczi, Mark Edmondson, Colin Fay, Ellis Hughes, Lars Kjeldgaard, Sean Lopp, Ben Marwick, Heather Nolis, Jacqueline Nolis, Hong Ooi, Karthik Ram, Noam Ross, Lori Shepherd, Péter Sólymos, Tyson Lee Swetnam, Nitesh Turaga, Charlotte Van Petegem, Jason Williams, Craig Willis, Nan Xiao

Abstract The Rocker Project provides widely used Docker images for R across different application scenarios. This article surveys downstream projects that build upon the Rocker Project images and presents the current state of R packages for managing Docker images and controlling containers. These use cases cover diverse topics such as package development, reproducible research, collaborative work, cloud-based data processing, and production deployment of services. The variety of applications demonstrates the power of the Rocker Project specifically and containerisation in general. Across the diverse ways to use containers, we identified common themes: reproducible environments, scalability and efficiency, and portability across clouds. We conclude that the current growth and diversification of use cases is likely to continue its positive impact, but see the need for consolidating the Rockerverse ecosystem of packages, developing common practices for applications, and exploring alternative containerisation software.

Introduction

The R community continues to grow. This can be seen in the number of new packages on CRAN, which is still on growing exponentially (Hornik et al., 2019), but also in the numbers of conferences, open educational resources, meetups, unconferences, and companies that are adopting R, as exemplified by the useR! conference series[1], the global growth of the R and R-Ladies user groups[2], or the foundation and impact of the R Consortium[3]. These trends cement the role of R as the *lingua franca* of statistics, data visualisation, and computational research. The last few years, coinciding with the rise of R, have also seen the rise of Docker as a general tool for distributing and deploying of server applications—in fact, Docker can be called the *lingua franca* of describing computing environments and packaging software. Combining both these topics, the *Rocker Project* (https://www.rocker-project.org/) provides Docker images with R (see the next section for more details). The considerable uptake and continued evolution of the Rocker Project has led to numerous projects that extend or build upon Rocker images, ranging from reproducible[4] research to production deployments. As such, this article presents what we may call the *Rockerverse* of projects across all development stages: early demonstrations, working prototypes, and mature products. We also introduce related activities that connect the R language and environment with other containerisation solutions. O

Resilience

Freedom

Simplicity

Major FaaS offerings

Azure Functions

Azure CLI
14G RAM limit
30 min timeout

AWS Lambda

AWS CLI
10G RAM limit
15 min timeout
API Gateway as extra

Google Cloud Functions

Gcloud CLI
4G RAM limit
9 min timeout

Repeated efforts + Vendor lock-in = Risk

OPENFAAS

ABOUT BLOG EBOOKS CONSULTING DOCS TEAM SUPPORT

27.1K VIEW ON GITHUB

Serverless Functions, Made Simple.

OpenFaaS® makes it simple to deploy both functions and existing code to Kubernetes

Anywhere

Avoid lock-in through the use of Docker. Run on any public or private cloud.

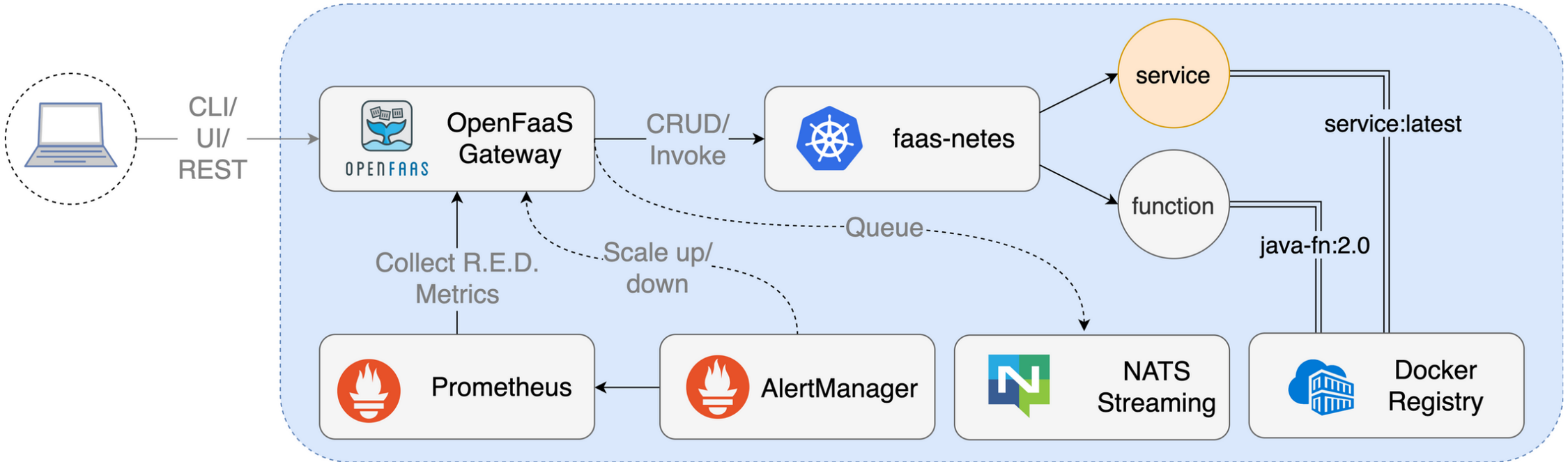
Any code

Build both microservices & functions in any language. Legacy code and binaries.

Any scale

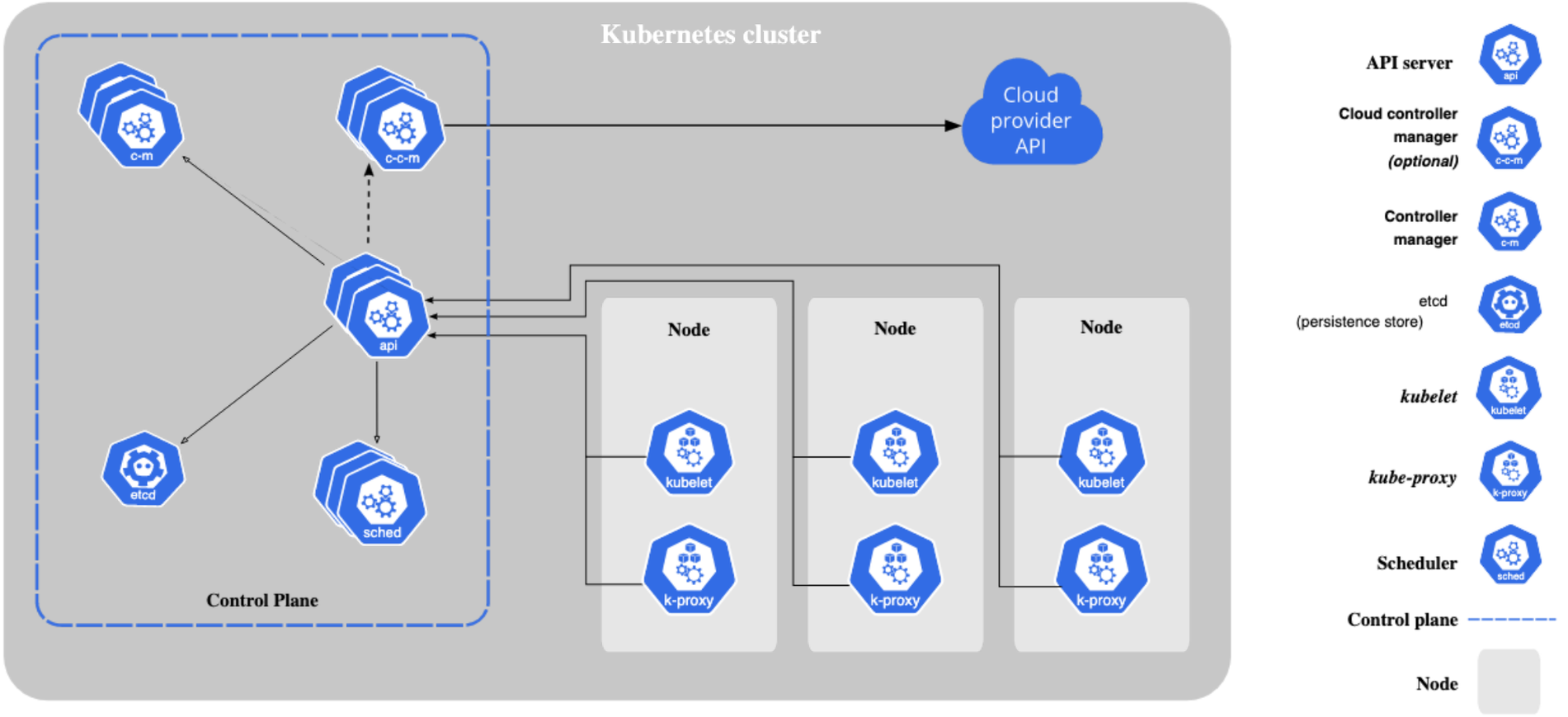
Auto-scale for demand or to zero when idle.

OpenFaaS.com



Focus on Kubernetes, by Alex Ellis

Kubernetes is a container orchestration engine for automating deployment, scaling, and management of containerized applications



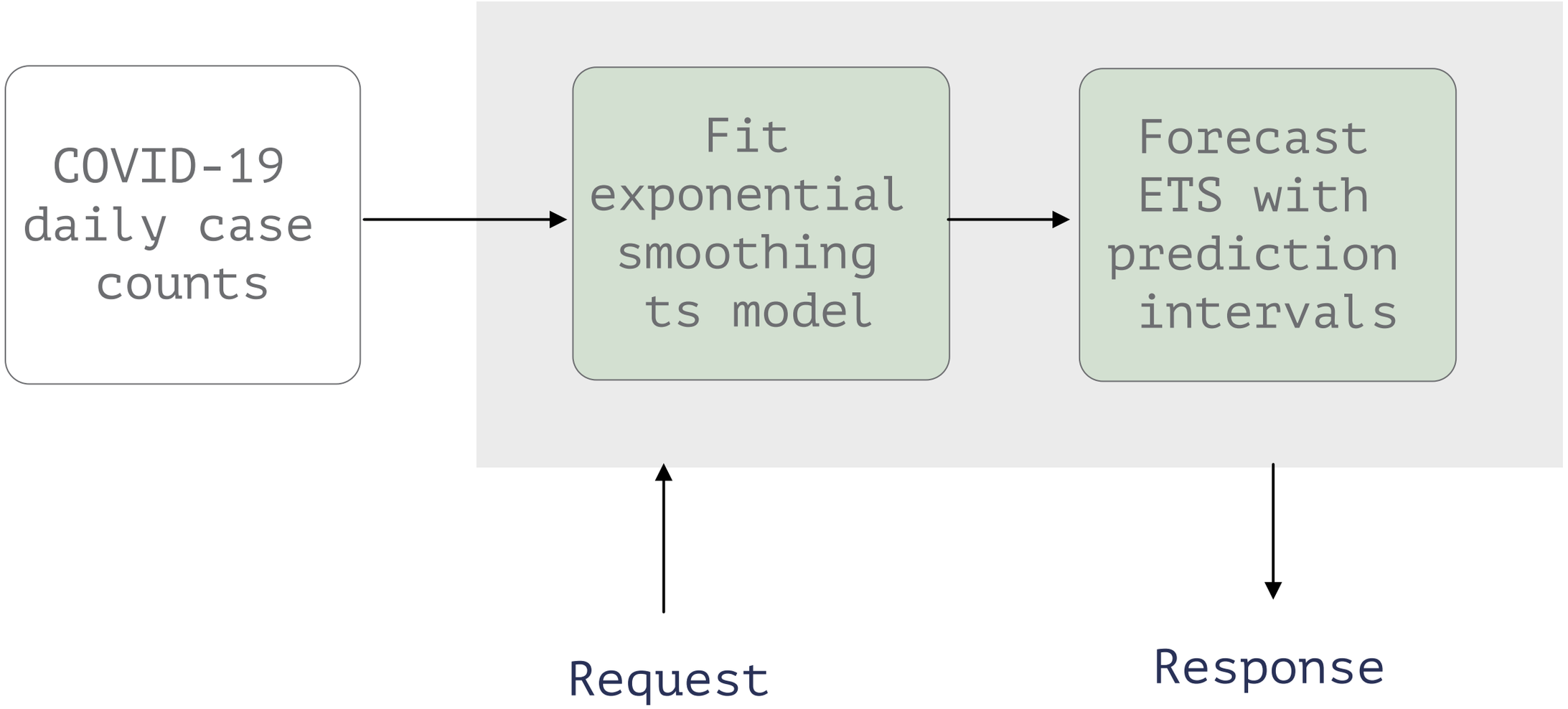
kubernetes.io

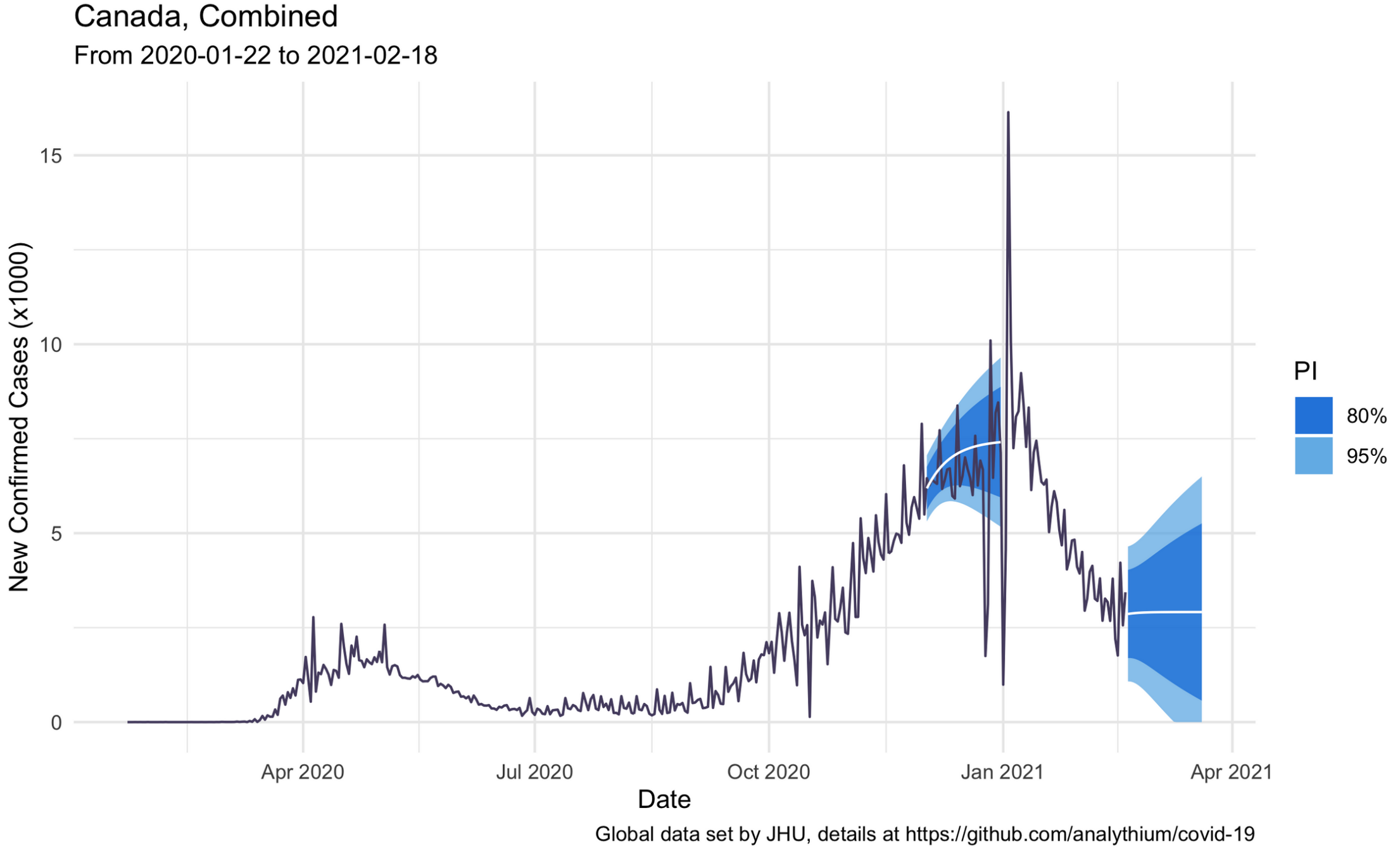
Resilience

Freedom

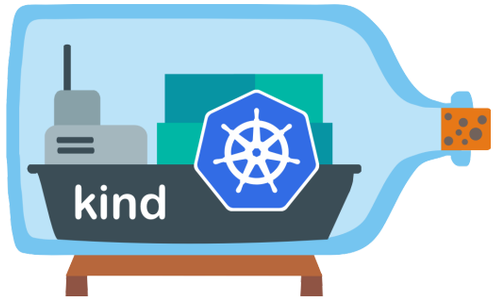
Simplicity

What we are going to do




Canada, Combined
From 2020-01-22 to 2021-02-18
New Confirmed Cases (x1000)
15
10
5
0
Apr 2020
Jul 2020
Oct 2020
Jan 2021
Apr 2021
Date
PI
80%
95%
Global data set by JHU, details at https://github.com/analythium/covid-19

Step 0: set up your cluster



Local cluster

KinD, minikube, k3s

Managed k8s

AWS, DO, Azure, GCP, etc

Lightweight faasd

Step 1: install faas-cli

Linux or macOS

Utility script with `curl`:

```
$ curl -sSL https://cli.openfaas.com | sudo -E sh
```

> The flag `-E` allows for any `http_proxy` environmental variables to be passed through to the installation bash script.

Non-root with curl downloads the binary into your current directory and will then print installation instructions:

```
$ curl -sSL https://cli.openfaas.com | sh
```

Via brew:

```
$ brew install faas-cli
```

Step 2: pull R templates

```
faas-cli template pull \
  https://github.com/analythium/openfaas-rstats-templates
```

R templates

	STDIO	httpuv	plumber	fiery	beakr	ambiorix
rocker/r-base	✓	✓	✓	✓	✓	✓
rocker/r-ubuntu	✓	✓	✓	✓	✓	✓
rhub/r-minimal	✓	✓	✓	✓	✓	✓

R templates

	STDIO	httpuv	plumber	fiery	beakr	ambiorix
rocker/r-base	✓	✓	✓	✓	✓	✓
rocker/r-ubuntu	✓	✓	✓	✓	✓	✓
rhub/r-minimal	✓	✓	✓	✓	✓	✓

Step 3: create new function

```
# Populate with your registry prefix or Docker Hub username
export OPENFAAS_PREFIX=""

# Populate with your OpenFaaS URL
export OPENFAAS_URL="http://174.138.114.98:8080"

faas-cli new --lang rstats-base-plumber \
  covid-forecast --prefix=$OPENFAAS_PREFIX
```

Step 4a: add handler

```
#* COVID
#* @get /
function(region, cases, window, last) {
  if (!missing(window))
    window <- as.numeric(window)
  covid_forecast(region, cases, window, last)
}
```

covid-forecast/handler.R

The R script loads the forecast package, defines the `covid_forecast` function with 4 arguments:

- `region` : a region slug value for the API endpoint in global data set
- `cases` : one of `"confirmed"` or `"deaths"` ,
- `window` : a positive integer giving the forecast horizon in days,
- `last` : last day (`"YYYY-MM-DD"` date format) of the time series to consider.

Step 4b: add dependencies

Add the forecast R package to the `covid-forecast/DESCRIPTION` file:

```
Package: COVID
Version: 0.0.1
Imports:
    forecast
Remotes:
SystemRequirements:
VersionedPackages:
```

Step 5: build + push + deploy

Now you can use `faas-cli up` to build, push, and deploy the COVID-19 forecast function to your OpenFaaS cluster:

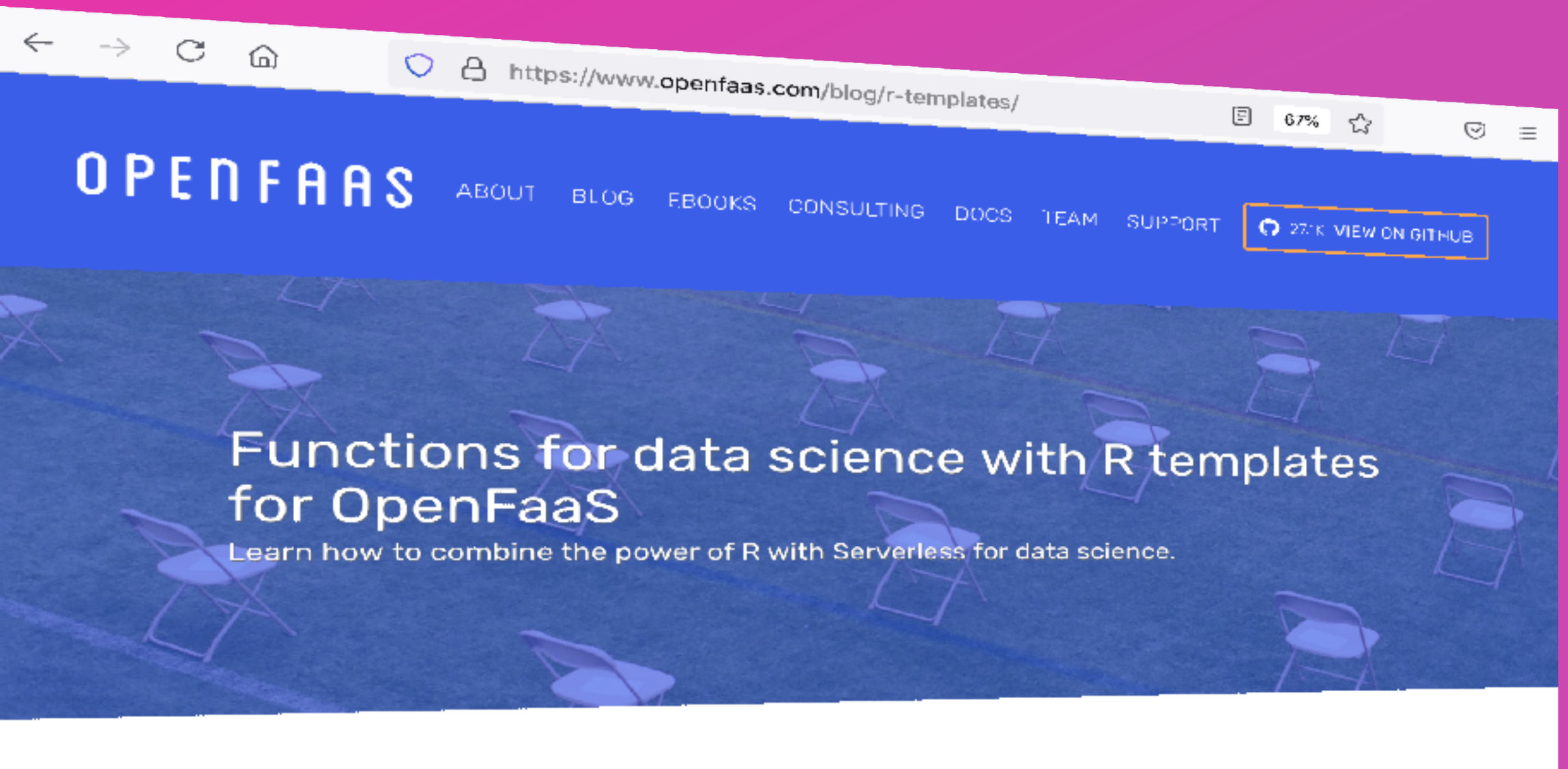
```
faas-cli up -f covid-forecast.yml
```

Step 6: invoke

```
$OPENFAAS_URL/function/covid-forecast?
region=canada-combined&window=4&last=2021-02-18
```


```
{
    "Date":["2021-02-19","2021-02-20","2021-02-21","2021-02-22"],
    "Point Forecast":[2861.5922,2871.8024,2879.9795,2886.5285],
    "Lo 80":[1694.8092,1695.4395,1686.1983,1667.6804],
    "Hi 80":[4028.3753,4048.1652,4073.7608,4105.3767],
    "Lo 95":[1077.1515,1072.7106,1054.2487,1022.4611],
    "Hi 95":[4646.0329,4670.8941,4705.7104,4750.596]
}
```

openfaas.com/blog/r-templates



Create and edit new function

```
faas-cli new hello \
  --lang rstats-base-plumber
```

handler.R

```
#* @post /
function(req) {
  sprintf("Hello %s!",
  fromJSON(paste(req$postBody)))
}
```

DESCRIPTION

```
Imports: jsonlite
Remotes:
SystemRequirements:
VersionedPackages:
```

R templates for OpenFaaS

https://github.com/analythium/openfaas-rstats-templates

Create and edit new function

```
faas-cli new hello \
  --lang rstats-base-plumber
```

handler.R

```
#* @post /
function(req) {
  sprintf("Hello %s!",
  fromJSON(paste(req$postBody)))
}
```

DESCRIPTION

```
Imports: jsonlite
Remotes:
SystemRequirements:
VersionedPackages:
```

Build, push, deploy

```
faas-cli up -f hello.yml
```

R templates for OpenFaaS

https://github.com/analythium/openfaas-rstats-templates

Create and edit new function

```
faas-cli new hello \
  --lang rstats-base-plumber
```

handler.R

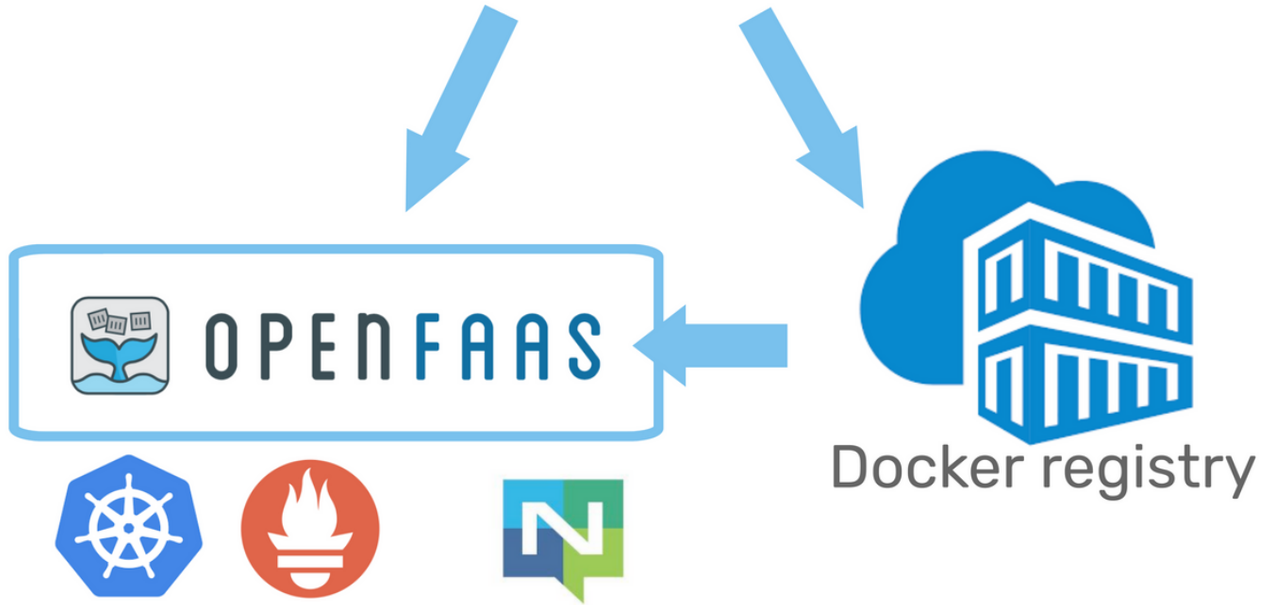
```
#* @post /
function(req) {
  sprintf("Hello %s!",
  fromJSON(paste(req$postBody)))
}
```

DESCRIPTION

```
Imports: jsonlite
Remotes:
SystemRequirements:
VersionedPackages:
```

Build, push, deploy

```
faas-cli up -f hello.yml
```



R templates for OpenFaaS

https://github.com/analythium/openfaas-rstats-templates

Create and edit new function

```
faas-cli new hello \
  --lang rstats-base-plumber
```

handler.R

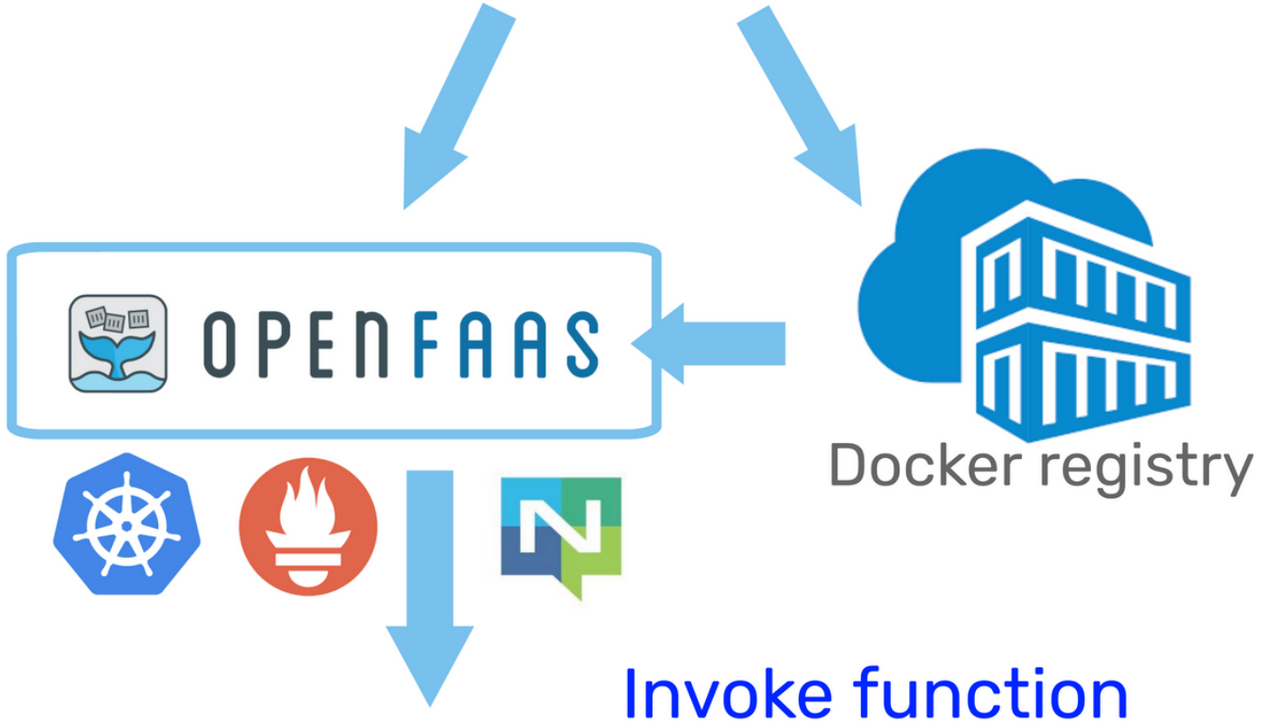
```
#* @post /
function(req) {
  sprintf("Hello %s!",
  fromJSON(paste(req$postBody)))
}
```

DESCRIPTION

```
Imports: jsonlite
Remotes:
SystemRequirements:
VersionedPackages:
```

Build, push, deploy

```
faas-cli up -f hello.yml
```



Invoke function

```
curl $OPENFAAS_URL/function/hello \
  -d '["World"]'
["Hello World!"]
```

R templates for OpenFaaS

https://github.com/analythium/openfaas-rstats-templates

Data science serverless-style with R and OpenFaaS

R templates

analythium/openfaas-rstats-templates

Examples

analythium/openfaas-rstats-examples

analythium.io