serveRless
computing for R

useR! 2019 – Toulouse
by Christoph Bodner & Thomas Laber
2
What does this buzzword actually mean?

Building a scalable and flexible pipeline to deploy R models

A solution architecture for Azure
What does this buzzword actually mean?

Building a scalable and flexible pipeline to deploy R models

A solution architecture for Azure
How can we build a cost effective data science pipeline that allows data scientists using R to easily put their models into production, that scales well and is cheap and easy to maintain?

“Rare picture of the fabled „eierlegende Wollmilch“

The Problem
What we want
a serverless data science architecture

get training data → Batch Training

Model storage

store models

get scoring data → read models

write results

Auto-Scaling

Batch Scoring  Realtime Scoring

docker pull

REST
What does this buzzword actually mean?

Building a scalable and flexible pipeline to deploy R models

A solution architecture for Azure
Just like wireless internet has wires somewhere, serverless architectures still have servers somewhere. What ‘serverless’ really means is that, as a developer you don’t have to think about those servers. You just focus on code.

Components of a serverless architecture

The Solution

“Just like wireless internet has wires somewhere, serverless architectures still have servers somewhere. What ‘serverless’ really means is that, as a developer you don’t have to think about those servers. You just focus on code.”

serverless.com
Why serverless?

The promise: Focus on coding, not maintenance

**NO ADMINISTRATION**
No server provisioning and maintenance is necessary. Hardware and OS are abstracted away.

**SCALE ON DEMAND**
Scaling is automatic and part of the service.

**PAY-PER-USE**
Billing is based on actual compute resources used. No compute used, no costs.

**FASTER TURNAROUND**
Spinning up new environments is quick and allows for faster experimentation.
## The Evolution of the Cloud

Cloud provider versus customer roles for managing cloud services

<table>
<thead>
<tr>
<th>Enterprise IT</th>
<th>Infrastructure as a Service</th>
<th>Platform as a Service</th>
<th>Functions as a Service</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Customer Managed</td>
<td>Provider Managed</td>
<td>Provider Managed</td>
</tr>
<tr>
<td>Applications</td>
<td>Applications</td>
<td>Applications</td>
<td>Applications</td>
</tr>
<tr>
<td>Scalability</td>
<td>Scalability</td>
<td>Scalability</td>
<td>Scalability</td>
</tr>
<tr>
<td>Security</td>
<td>Security</td>
<td>Security</td>
<td>Security</td>
</tr>
<tr>
<td>OS</td>
<td>OS</td>
<td>OS</td>
<td>OS</td>
</tr>
<tr>
<td>Virtualization</td>
<td>Virtualization</td>
<td>Virtualization</td>
<td>Virtualization</td>
</tr>
<tr>
<td>Servers</td>
<td>Servers</td>
<td>Servers</td>
<td>Servers</td>
</tr>
<tr>
<td>Storage</td>
<td>Storage</td>
<td>Storage</td>
<td>Storage</td>
</tr>
<tr>
<td>Networking</td>
<td>Networking</td>
<td>Networking</td>
<td>Networking</td>
</tr>
<tr>
<td>Data Centers</td>
<td>Data Centers</td>
<td>Data Centers</td>
<td>Data Centers</td>
</tr>
</tbody>
</table>

- **Entreprise IT**
  - legacy IT

- **Infrastructure as a Service**
  - Virtual Machines

- **Platform as a Service**
  - Containers

- **Functions as a Service**
  - Functions
Cost Comparison

Serverless can be cheap, but depends on work load

Big lock-in potential!
What does this buzzword actually mean?

Serverless
Building a scalable and flexible pipeline to deploy R models

01 The Problem

02 Serverless

03 Architecture
A solution architecture for Azure
Two Use Cases
Model training and scoring have different architecture requirements

TRAINING
• Usually long running tasks
• Resource intensive
• Mostly in batch mode

SCORING
• Mostly short running tasks
• Resource usage low
• Either adhoc or on schedule

OUR FOCUS TODAY
Serverless Options

We primarily looked at the following options:

- AWS Lambda
- Azure Functions
- Azure Container Instances
Requirements

Many ways to realize serverless scoring architecture with different pros and cons

- Trigger
- Deploy Resources
- Load model
- Serve Model Score

Must support at least time/http trigger

Loading from blobs

Custom runtime support necessary

Optionally serving scores with HTTP
A function can use up to 5 layers at a time. The total unzipped size of the function and all layers can’t exceed the unzipped deployment package size limit of 250MB.

Compiled packages can be a headache...

Philipp Schirmer
Function as a Service

Azure Functions

- C#
- Java
- .NET
- .NET Core

functions V2 runtime

language worker process

host process

function code

gRPC

modern open source high performance RPC framework

Neal Fultz

Protocol Buffers

Google's language-neutral, platform-neutral, extensible mechanism for serializing structured data

Dirk Eddelbuettel
Why Azure Container?

Container give us maximum flexibility regarding runtime and reduce vendor lock-in.

**PROS**

- Supports arbitrary runtimes
- No problems with compiled libraries
- Lots of supported triggers in combination with logic apps
- Low vendor lock-in
- Pay-as-you-go

**CONS**

- More setup involved compared to FaaS such as AWS Lambda
- Higher startup times compared to FaaS depending on Image
01 Logic App
Logic App implements trigger (time/event) and spawns Container Instances

02 Container Instances
Container Instances pulls Docker image from Azure Container Registry or other Container Registry

03 Container Registry
Model scoring code in Docker image gets pulled to ACI

04 Blob Store
Load serialized models for scoring from blob storage

Azure Container + Logic App
Our setup currently looks like this
Logic App
A serverless workflow orchestration tool with GUI for prototyping

LOGIC APP DESIGNER

1. When a HTTP request is received
2. Create container group (Preview)
3. Until
4. Delete container group (Preview)

LOGIC APP TEMPLATING

```
{
    "$connections": {
        "value": [
            {
                "connectionId": "subscriptions/subscription/resourceGroups/serverless/providers/Microsoft.Web/appServices/logicAppAndHttp/sites/logicApp",
                "connectionName": "acl",
                "id": "subscriptions/subscription/providers/Microsoft.Web/appServices/logicAppAndHttp/sites/logicApp/appsettings",
                "name": "connection-1"
            }
        ]
    },
    "$schema": "https://schema.management.azure.com/providers/Microsoft.Logic/schemas/2016-06-01/WorkflowLocations/azure/",
    "definition": {
        "actions": {
            "Create_container_group": {
                "inputs": {
                    "body": {
                        "location": "westeurope",
                        "properties": {
                        }}
                    },
```
Scoring Workflow
Template workflow for a wide range of scenarios

Specify trigger type (time, HTTP, email, etc)

Create container resources based on spec (cpu + RAM + nodes)

Check if container group is spawned successfully

Delete container after work is done
serveRless Package

We want to build a package to help automate this setup

IDEA

- Build an R package that allows R users to deploy their code in a serverless setup

STATUS

- [x] Build prototype to test setup
- [ ] Build Rstudio Addin
- [ ] R Package serveRless

Many thanks to Hong Ooi for his awesome work supporting R in Azure!
Questions?
Thank you for your attention!

Feel free to reach out to us:

linkedin.com/in/christoph-bodner
linkedin.com/in/thomas-laber
## What now?

### VMs vs containers vs functions

<table>
<thead>
<tr>
<th></th>
<th>Virtual Machines</th>
<th>Containers</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Unit of Scale</strong></td>
<td>machine</td>
<td>application</td>
<td>function</td>
</tr>
<tr>
<td><strong>Abstraction</strong></td>
<td>hardware</td>
<td>operation system</td>
<td>language runtime</td>
</tr>
<tr>
<td><strong>Packaging</strong></td>
<td>image</td>
<td>container file</td>
<td>code</td>
</tr>
<tr>
<td><strong>Configure</strong></td>
<td>machine, storage, network, OS</td>
<td>servers, applications, scaling</td>
<td>run code when needed</td>
</tr>
<tr>
<td><strong>Execution</strong></td>
<td>multi-threaded, multi-task</td>
<td>multi-threaded, single-task</td>
<td>single-threaded, single-task</td>
</tr>
<tr>
<td><strong>Runtime</strong></td>
<td>hours to months</td>
<td>minutes to days</td>
<td>microseconds to seconds</td>
</tr>
<tr>
<td><strong>Unit of Cost</strong></td>
<td>per VM per hour</td>
<td>per container per hour</td>
<td>per memory/second per request</td>
</tr>
<tr>
<td><strong>Amazon</strong></td>
<td>EC2</td>
<td>Fargate</td>
<td>Lambda</td>
</tr>
<tr>
<td><strong>Azure</strong></td>
<td>Azure VM</td>
<td>Container Instances</td>
<td>Azure Functions</td>
</tr>
<tr>
<td><strong>Google</strong></td>
<td>Google Compute Engine</td>
<td>Google Kubernetes</td>
<td>Cloud Functions</td>
</tr>
</tbody>
</table>