Tradition trifft Moderne:
PHP im Kontext von Serverless

Serverless Computing mit PHP

Schenkt man den Cloudarchitekturpropheten Glauben, hat die klassische Webanwendung bald ausgedient und tritt nur noch in Legacy-Systemen in Erscheinung. Die Sprache PHP hat – gemessen an den Zeiträumen des Internets – eine lange und erfolgreiche Geschichte hinter sich und wird produktiv in unzähligen Webanwendungen eingesetzt. Innerhalb der als „Zukunft der Softwareentwicklung" [1] bezeichneten Serverless-Architektur konnte PHP jedoch bislang keinen führenden Platz erobern. Das Bref-Projekt schickt sich an, das im Rahmen von AWS Lambda zu ändern.

von Ralf Geschke

„PHP ist eine serverseitig interpretierte, in HTML eingebettete Skriptsprache zur Entwicklung dynamischer Webseiten …" – so oder so ähnlich lautete der erste Satz der einführenden Zusammenfassung auf den kurz vor der Jahrtausendwende entstandenen deutschsprachigen Informations- und Portalseiten zum Thema PHP. Aus einer Sammlung von Skripten, die von Rasmus Lerdorf zunächst für die Verarbeitung von HTML-Formularen geschrieben worden war (PHP/FI), wurde innerhalb kurzer Zeit eine Sprache, die eine effiziente Entwicklung von datenbankgetriebenen Webseiten ermöglichte. Anstatt auf Perl oder Java zu setzen, wurde PHP in der Frühzeit des Webs und der aufstrebenden New Economy immer häufiger genutzt, um webbasierte Anwendungen zu erstellen. Nach einigen Evolutions- und manchen Revolutionsschritten ist die Sprache erwachsen geworden und steht aktuell in der Version 7.3 bereit. PHP hat damit nicht nur die ersten Jahre und die erste Programmierergeneration des Webs geprägt, sondern wurde im Lauf der Zeit zur meistverwendeten serverseitigen Programmiersprache [2]. Diese Position hält PHP auch heute noch – beispielsweise hat WordPress mit über 60 Prozent den größten Marktanteil im CMS-Bereich inne [3]. PHP wird von den meisten Webhostern standardmäßig angeboten, insofern hat sich an der Relevanz in den letzten Jahren kaum etwas verändert. Natürlich programmiert heute – im besten Fall – niemand mehr mit PHP auf die ursprüngliche Art und Weise. Analog zur Weiterentwicklung der Sprache hat sich auch der Programmierstil geändert. Inzwischen bestim-

men Frameworks wie Symfony oder Laravel die Szene, anstelle von selbsterstellten Skripten werden Standardarchitekturkonzepte adaptiert, um die Entwicklung effizienter und die jeweilige Anwendung modular erweiterbar und wartbar zu machen.

Beim Serverless-Dienst AWS Lambda fehlt PHP jedoch bislang in der Liste der von Amazon angebotenen Laufzeitumgebungen. Unterstützt werden standardmäßig JavaScript mit Node.js, Python, Ruby, Java, Go und .NET mit C# [4]. Zur Verminderung eines Sprachenschlagabtauschs und Framework-Fehden hat Amazon im November 2018 das Lambda Runtime API veröffentlicht, das es erlaubt, weitere Laufzeitumgebungen einzusetzen [5]. An dieser Stelle tritt das Projekt Bref auf den Plan [6]. Bref ist Open Source und wird federführend vom Softwarearchitekten und -consultant Matthieu Napoli und dessen Unternehmen entwickelt [7]. Bref bezeichnet sich als Framework und besteht aus mehreren Komponenten, die die Nutzung von PHP auf AWS Lambda ermöglichen. Neben den reinen Laufzeitumgebungen für PHP gehört ein CLI-Tool für Scaffolding und Deployment dazu, ebenso wird die Dokumentation explizit als ein Bestandteil von Bref genannt.

PHP-Entwicklung mit Bref

Waren für Bref in den Versionen vor 0.5.0 noch das AWS-CLI-Tool sowie das AWS SAM (Serverless Application Model) CLI notwendig, stützt sich Bref inzwischen auf das Serverless Framework, ein Open-Source-Build- und -Deploy-System, das eine einheitliche Schnittstelle zu AWS und weiteren Cloudanbietern, darunter die Google Cloud Platform und Microsoft Azure, anbietet [8]. Für das Serverless Framework muss npm, der Paketmanager für Node.js, installiert sein. Für Bref werden PHP in der CLI-Variante inklusive cURL-, XML- und ZIP-Libraries sowie Composer, das projektbezogene Paket- und Abhängigkeitsmanagementtool für PHP, benötigt. Auf eine Anleitung zur Installation der genannten Tools wird an dieser Stelle verzichtet. Die Dokumentationsseiten der Tools stellen ausführliche Hinweise zur Inbetriebnahme auf unterschiedlichen Betriebssystemen zur Verfügung [9], [10].

Für die globale Installation des Serverless Frameworks ist folgendes Kommando zuständig:

```
$ sudo npm install -g serverless
```

Nach einigen Downloads, der Auflösung von Abhängigkeiten, Installation Tausender Dateien und obligatorischer Ausgabe von npm-Warnungen ist das Serverless-CLI einsatzbereit. Wird *serverless* ohne einen weiteren Parameter gestartet, findet man sich in einem Dialog wieder, um ein Projekt zu erstellen. Eine Liste der zur Verfügung stehenden Kommandos erhält man mit der *help*-Option, die je nach Kontext erweiterte Informationen ausgibt:

```
$ serverless --help
```

Damit das Serverless Framework Zugriff auf die benötigten AWS-Ressourcen erhält, müssen die Zugriffsdaten in Form von Key und Secret bekanntgegeben werden. Des Weiteren muss der Provider genannt werden. Das notwendige Kommando lautet für AWS wie folgt:

```
$ serverless config credentials --provider aws --key <key> --secret <secret>
```

Für den Zugriff auf die Services von AWS müssen im IAM (Identity and Access Management) Service die entsprechenden Rechte angelegt werden. Neben den AWS-verwalteten Richtlinien *AWSLambdaFullAccess* und *AmazonAPIGatewayAdministrator* wird mindestens Zugriff auf die Services *CloudFormation* und *IAM* benötigt. Die so erstellte Richtlinie ist in einem Gist-Eintrag bei GitHub zu finden [11]. Bei Verwendung weiterer AWS-Dienste sind die Zugriffsrechte entsprechend zu erweitern. Damit sind die Vorbereitungen abgeschlossen und Bref kann im Projektverzeichnis installiert werden:

```
~/$ mkdir httptest && cd httptest
~/httptest$ composer require bref/bref
```

Nach der erfolgreichen Installation lässt sich das Bref-CLI aufrufen:

```
~/httptest$ vendor/bin/bref
```

Ohne Angabe von Parametern erfolgt die Ausgabe der Optionen und verfügbaren Kommandos.

PHP-Laufzeitumgebungen

Bevor ein Projekt per *bref init* eingerichtet wird, sollte eine Entscheidung über die Funktionalität und damit implizit über die zu benutzende Runtime getroffen werden. Bref stellt hier drei Varianten zur Verfügung:

Bei der **PHP function** wird eine Bref-interne Funktion namens *lambda()* genutzt, die den Funktions-Handlern in anderen Sprachen entspricht. Der *lambda()*-Funktion wird eine PHP-Funktion in Form eines Callbacks oder einer anonymen Funktion übergeben. Die Funktion erhält von der Runtime einen Parameter *$event* und optional ein *$context*-Argument. In *$event* werden Informationen zum Ereignis übergeben, das den Funktionsaufruf ausgelöst hat. Die Angaben in *$context* sind analog zum Context-Objekt in anderen Umgebungen, etwa bei Node.js [12]. Die Rückgabe ist ebenfalls optional, muss jedoch, falls vorhanden, als JSON serialisierbar sein. Der Funktionsaufruf lässt sich in weitere AWS Services integrieren, wodurch eine Reaktion auf unterschiedliche AWS-Events möglich ist [13].

Im Vergleich dazu entspricht die Variante **HTTP application** der klassischen Ausführung einer Webanwendung. In der Laufzeitumgebung wird PHP-FPM (FastCGI Process Manager) eingesetzt, wie man es etwa vom Betrieb mit dem Webserver NGINX kennt. Die Rolle des Handlers übernimmt eine PHP-Datei, die in der Konfiguration angegeben werden muss.

Die letzte Variante ist **Console**. Dabei können PHP-Anwendungen auf der Konsole ausgeführt werden, um beispielsweise die Symfony Console oder Laravel Artisan zu nutzen.

Alle Runtimes und ihre Eigenschaften sind in der Bref-Dokumentation ausführlich beschrieben [14]. Da die HTTP Application einer traditionellen PHP-Anwendung am nächsten kommt, bezieht sich das nachfolgende Beispiel darauf.

Hello, (Serverless PHP) World!

Im ersten Schritt wird eine Anwendung durch den Aufruf von *bref init* generiert:

```
~/httptest$ vendor/bin/bref init
```

Bref fragt nach der Art der Anwendung, die Angabe von 1 wählt eine HTTP Application. Anschließend generiert Bref zwei Dateien: *index.php* und *serverless.yaml*. Die PHP-Datei besteht aus einem *echo*-Konstrukt, das einen String ausgibt:

```
<?php
echo 'Hello world!';
```

Interessanter ist die Datei *serverless.yml*. Mit diesem Template werden die Architektur und die notwendigen Ressourcen festgelegt. Das Serverless Framework greift dabei auf den AWS-Dienst CloudFormation zurück, der umfassende Möglichkeiten bietet, eine Serverless-Infrastruktur aufzubauen und zu deployen. Die Abstraktionsschicht des Serverless Frameworks soll die Komplexität von CloudFormation verringern, ohne dessen Möglichkeiten einzuschränken. Im einfachsten Fall besteht *serverless.yml* nur aus wenigen Zeilen, die Referenz in der Dokumentation stellt alle verfügbaren Optionen vor [15]. Das Beispiel in Listing 1 zeigt die von Bref generierte und geringfügig angepasste Datei *serverless.yml*.

Listing 1

```
service: httptestsrv

provider:
  name: aws
  region: eu-central-1
  runtime: provided
  stage: prod

plugins:
  - ./vendor/bref/bref

functions:
  httptest:
    handler: index.php
    description: ''
    timeout: 30 # in seconds (API Gateway has a timeout of 30 seconds)
    layers:
      - ${bref:layer.php-73-fpm}
    events:
      - http: 'ANY /'
      - http: 'ANY /{proxy+}'
```

Der Service ist der Projektname im Rahmen des Serverless Frameworks. Er findet sich nach dem Deployment im Namen des CloudFormation-Stacks und der Lambda-Funktion sowie in nahezu allen bereitgestellten AWS-Ressourcen wieder. Ein Stack ist die vollständige Beschreibung einer Sammlung von AWS-Ressourcen und definiert die Anwendung. Beim Deployment sorgt AWS CloudFormation für die Einrichtung der Ressourcen, kümmert sich um die notwendigen Rechte und löst Abhängigkeiten auf.

Anschließend finden sich Properties zur Konfiguration des Providers. Darin wird die AWS-Region gewählt, deren Standardwert *us-east-1* auf den Eintrag von Frankfurt, *eu-central-1*, geändert wurde. Die Eigenschaft *runtime* muss beibehalten werden und bezieht sich auf die von Bref bereitgestellten Runtimes in der Option *layers* weiter unten in der Datei. Die Angabe der Bereitstellungsumgebung (*stage*) ist optional und besitzt per Default den Wert *dev*. Abgesehen von der Bedeutung im Entwicklungsprozess hat eine Anpassung auch konkrete Auswirkungen auf die Bezeichner der Ressourcen der AWS-Komponenten. Das Serverless Framework erzeugt etwa den Namen des CloudFormation-Stacks aus dem Namen des Service, ergänzt durch Stage: *<service>-<stage>*, im Beispiel *httptestsrv-prod*. Analog dazu heißt die einzige Funktion aus dem Beispiel *httptestsrv-prod-httptest*.

Die Definition des Handlers legt fest, welche PHP-Datei bei einem HTTP Request aufgerufen wird, im Beispiel ist das *index.php* im aktuellen Verzeichnis. Die Angabe des Layers wählt eine Laufzeitumgebung, die von Bref in der jeweiligen Region zur Verfügung gestellt wird. Bref bietet für die aktuellen Versionen PHP 7.2 und PHP 7.3 jeweils für die drei unterschiedlichen Runtime-Varianten (Function, HTTP, Console) in einer Reihe von AWS-Regionen an [16].

Der Bereich *events* legt fest, bei welchen Ereignissen der Aufruf der AWS-Lambda-Funktion erfolgen soll. Für einen HTTP Request, der durch den AWS-Service API Gateway bedient wird, richtet *http* die entsprechende Ressource unter dem angegebenen Pfad ein. Anstatt des Catch-all-Routings des Beispiels können auch komplexere Konfigurationen realisiert werden, etwa durch Festlegung einzelner Pfade oder Einschränkung auf einzelne HTTP-Methoden wie *GET*, *POST*, *PUT* usw. [17]. Dem so generierten URL, auch API-Endpunkt genannt, wird die Stage-Angabe am Ende des Pfads hinzugefügt.

Deployment auf AWS

Für das Deployment ist dank des Serverless Frameworks nur ein Kommando notwendig:

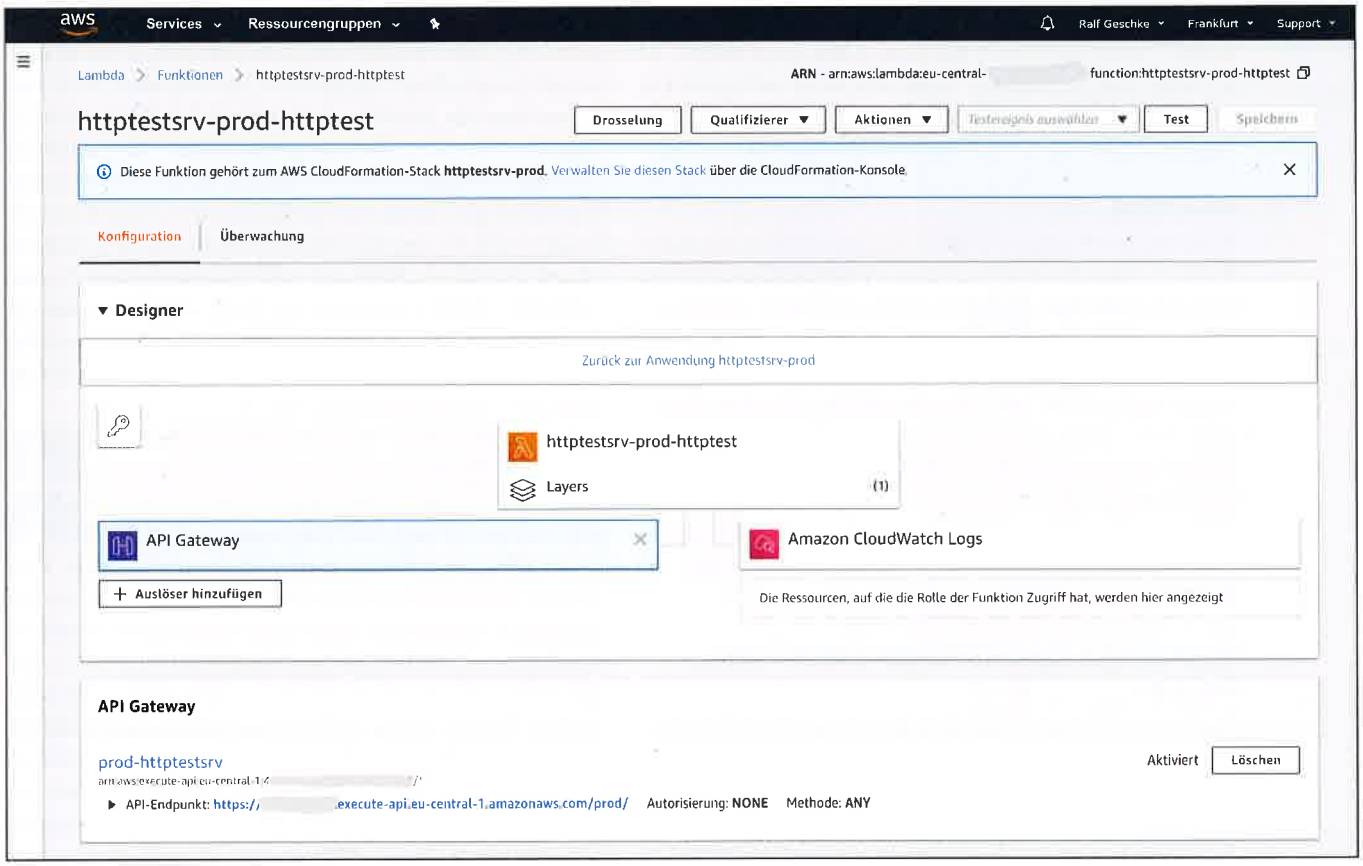

Abb. 1: Architektur einer Lambda-Funktion im AWS UI

```
~/httptest$ serverless deploy
```

Daraufhin werden die Schritte ausgeführt, die vor Bref 0.5.0 mit den AWS-eigenen CLI-Tools erledigt werden mussten. Im Einzelnen handelt es sich um:

- Erzeugen des CloudFormation-Stacks, initial für die Bereitstellung eines Speicherbereichs (Bucket) beim Objektspeicher S3
- Packen der Quelldateien als ZIP-File
- Upload des Funktionscodes in den S3-Bucket
- Hinzufügen von Ressourcen wie Referenz zum ZIP-File, IAM-Rollen, Events usw.
- Update des CloudFormation-Stacks, wobei die Ressourcen aus dem letzten Schritt angelegt bzw. aktualisiert werden

Alle Aktionen werden nur bei Bedarf ausgeführt, vorhandene Ressourcen werden, falls möglich, erneut genutzt. Um Platz zu sparen, ist es empfehlenswert, Composer vor dem Deployment mit den Optionen *--optimize-autoloader* und *--no-dev* aufzurufen. Letztere verhindert, dass Libraries installiert werden, die nur während der Entwicklung notwendig sind.

Nachdem das Deployment erfolgreich abgeschlossen ist, werden Informationen über den Service ausgegeben. Eine der wichtigsten dürfte die Angabe des API-Endpunkts sein, denn ein HTTP Request auf diesen URL sollte mit dem beliebten Gruß antworten.

An dieser Stelle lohnt ein Blick auf das AWS-Web-UI (**Abb. 1**) zur Erkundung des CloudFormation-Stacks, der Lambda-Funktion oder des Überwachungsdiensts CloudWatch. Wird auf die Standardfehlerausgabe *stderr* geschrieben, sind diese Angaben als Teil der Logs erreichbar, die bei CloudWatch im Rahmen der Protokolle für jeden einzelnen Request zur Verfügung stehen. Diese Einträge können auch mit dem Serverless CLI abgerufen werden:

```
~/httptest$ serverless logs --function httptest
```

Ebenso lassen sich Angaben zum Service selbst, insbesondere zum API-Endpunkt, jederzeit über das CLI ausgeben:

```
~/httptest$ serverless info
```

PHP-Anwendungen, die über eine einzeilige Ausgabe hinausgehen, möchte man meist lokal testen. Dazu kann der im PHP CLI integrierte Webserver dienen:

```
~/httptest$ php -S localhost:3000
```

Anstatt *localhost* kann ebenso der DNS-Name oder die IP-Adresse des Entwicklungsservers verwendet werden. Da sich die lokale Umgebung und die AWS Runtime voneinander unterscheiden können, sollte vor dem Deployment die Ausführung in der Bref-Laufzeitumgebung

stattfinden. Diese Möglichkeit wurde von Bref in Versionen kleiner als 0.5.0 mit dem AWS-Tool SAM CLI bereitgestellt. Leider gibt es aktuell keine adäquate Lösung, jedoch arbeiten die Autoren von Bref daran, eigene Docker Images mit den Runtimes bereitzustellen [18].

Schichtung

Interessant ist auch ein Blick in den Aufbau und den Build-Prozess der Bref Runtimes [19]. Als Basis wird ein von Amazon bereitgestelltes Linux Image verwendet. Anhand eines ersten Dockerfiles werden Compiler und Entwicklungstools installiert. In einem weiteren Docker Image finden PHP und dessen Abhängigkeiten (Libraries) Platz. Zum Schluss wird ein Container aufgrund des zuvor erzeugten PHP Image gestartet, aus dem die für die Runtime notwendigen Dateien extrahiert werden. Ein Layer ist damit nichts Anderes als ein Archiv mit Dateien, die zusammen mit dem Basis-Image die Ausführung von PHP ermöglichen.

Erweiterungen, Einschränkungen und ein reales Beispiel

Bei den verfügbaren PHP Extensions hat Bref eine Auswahl unter den häufig genutzten getroffen und sie in den Build-Prozess mitaufgenommen. Der Verwendung eigener bzw. weiterer PHP Extensions steht jedoch nichts im Weg. Dazu können weitere Layer auf der Basis von Bref erstellt werden, die die benötigten Libraries und Extensions beinhalten.

Einige Extensions sind zwar in den Bref Runtimes vorhanden, aber nicht aktiviert. Das kann durch die Nutzung einer eigenen *php.ini*-Datei zur Konfiguration geändert werden. Bref geht im Standardfall davon aus, dass sich im Projektverzeichnis eine Datei *php.ini* im Unterverzeichnis *php/conf.d/* befindet [20].

Als weiteres Beispiel soll ein Kontaktformular dienen. Aus statischen HTML-Seiten werden per JavaScript Eingaben an den Server geschickt und es wird dessen JSON-Antwort ausgewertet. Würde die Anwendung mehr Businesslogik beinhalten, könnte man sie als Single Page Application bezeichnen. Auf dem Server werden die Eingaben geprüft und Googles ReCaptcha Service wird angesprochen. Anschließend wird das Feedback an die Mailadresse des Websitebetreibers geschickt. Die Beispielanwendung läuft seit einiger Zeit in der Serverless-Umgebung der Alibaba Cloud (Kasten: „PHP bei AWS-Konkurrenten") und sollte für AWS Lambda und Bref adaptiert werden. Die Anpassungen für die Lambda-Umgebung werden im kommentierten Quellcode beschrieben, der auf GitHub zur Verfügung steht [30].

Bereits bei einem einfachen Beispiel zeigt sich, dass eine Modifikation von bestehenden Anwendungen in den meisten Fällen notwendig sein wird. Die Gründe liegen weniger in den PHP Binaries der Bref Runtime als in der Cloudumgebung von AWS Lambda selbst. Eine weitere Einschränkung ist das Read-only-Filesystem der Lambda-Umgebung, einzig das Verzeichnis */tmp/* ist beschreibbar, die Inhalte werden jedoch nicht zwischen den Instanzen geteilt. Wird ein Lambda Service in einer Virtual Private Cloud (VPC) eingerichtet, um etwa eine Datenbank in einer gesicherten Umgebung zu nutzen, können einige Sekunden bis zum Start der Funktion vergehen. Des Weiteren können externe Dienste aus einer VPC heraus nicht erreicht werden, Googles ReCaptcha wäre somit nicht nutzbar. Diese Eigenheiten sind jedoch nicht Bref-spezifisch, sondern gelten allgemein für alle AWS-Lambda-Umgebungen.

Ab in die Cloud

Insgesamt gesehen kann Bref überzeugen, sofern ein passendes Anwendungsszenario vorhanden ist. Es erscheint wenig sinnvoll, zu versuchen, eine monolithische PHP-Anwendung wie WordPress mit seinen unzähligen Themes und Plug-ins in einer serverlosen Umgebung zu betreiben. Dafür ist AWS Lambda letztlich auch nicht

PHP bei AWS-Konkurrenten

Unter den Cloudanbietern besitzt AWS den größten Marktanteil. Doch die Konkurrenz schläft nicht und hat Amazon bezüglich des jährlichen Wachstums sogar überholt [21]. Wie ist die Unterstützung von PHP bei den Verfolgern?

Microsoft Azures Serverless-Dienst heißt Azure Functions. Zwar taucht PHP in der Übersichtsseite zu Azure Functions auf, in der Liste der unterstützten Sprachen sind für die aktuelle Runtime-Version 2.x neben den Microsoft-eigenen Entwicklungen C#, F# und PowerShell nur JavaScript mit Node.js, Java und Python als allgemein verfügbar gekennzeichnet [22], [23]. PHP wird für die ältere Runtime-Version 1.x mit dem Hinweis „experimentell" angegeben, was eine zukünftige Unterstützung fraglich erscheinen lässt. Einen Weg, um PHP in Microsofts .NET-Welt zu bringen, könnte das PeachPie-Projekt darstellen [24].

Die Google Cloud Platform (GCP) stellt mit den Cloud Functions einen Serverless-Dienst bereit. Auch dort finden sich JavaScript mit unterschiedlichen Node.js-Versionen sowie Python und – nicht weiter verwunderlich – Googles Go in der Liste der Runtimes [25]. PHP wird nicht unterstützt, ebenfalls existiert zum aktuellen Zeitpunkt keine Möglichkeit der Nutzung einer Custom Runtime.

Ganz anders stellt sich die Situation beim aufstrebenden Cloudanbieter aus China, Alibaba Cloud, dar. Dessen Serverless-Dienst namens Function Compute bietet neben den gewohnten Umgebungen für JavaScript mit Node.js, Java und Python offiziell Unterstützung für PHP an [26]. Die Entwickler der Alibaba Cloud haben dafür eine Runtime auf Basis der ReactPHP-Library für Event-gestützte Programmierung erstellt [27], [28]. Als zentraler Einstiegspunkt dient eine Handler-Funktion, die der Variante PHP function von Bref ähnlich ist. Auch können PHP Frameworks wie Symfony mit ReactPHP verwendet werden, wobei zumindest eine Anpassung von Request und Response vonnöten ist [29].

gedacht. Eine erste Einschätzung über den Nutzen von Bref unter Berücksichtigung des aktuellen Entwicklungsstands gibt die „Maturity Matrix" aus der Bref-Dokumentation [31]. Des Weiteren muss beachtet werden, dass AWS Lambda nicht allein steht, sondern in die sehr umfassende Cloud-Service-Infrastruktur von Amazon eingebettet ist. Für das Kontaktformularbeispiel werden inklusive Deployment und Logging insgesamt acht unterschiedliche AWS-Dienste in Anspruch genommen. Die damit einhergehende Komplexität will erst einmal beherrscht werden – dabei helfen Bref und das Serverless Framework. Angesichts der Qualität, die Bref in der bisherigen Entwicklungszeit erreicht hat, wäre es auf alle Fälle wünschenswert, wenn Amazon PHP mit Bref in die Liste der von AWS Lambda unterstützten Sprachen aufnimmt und einen derartigen Hinweis an prominenter Stelle zugänglich macht.

Ralf Geschke ist Softwareentwickler und PHP-Veteran, genauso wie Nerd und DevOps-Mensch. Er beschäftigt sich gerne mit neuen Technologien (aka „heißem Scheiß") rund um Buzzwords wie Cloud-Native, Docker, Kubernetes und Co., lässt Open-Source-Seed-Tomaten wachsen und verbraucht viel zu viel grünen Strom für die Server im Keller.

Links & Literatur

[1] https://jaxenter.de/serverless-kubernetes-faas-eroeffnung-82294
[2] https://w3techs.com/technologies/overview/programming_language/all
[3] https://de.statista.com/statistik/daten/studie/320670/umfrage/marktanteile-der-content-management-systeme-cms-weltweit/
[4] https://docs.aws.amazon.com/de_de/lambda/latest/dg/lambda-runtimes.html
[5] https://aws.amazon.com/de/blogs/aws/new-for-aws-lambda-use-any-programming-language-and-share-common-components/
[6] https://bref.sh
[7] https://null.tc
[8] https://serverless.com/framework/
[9] https://nodejs.org/en/download/
[10] https://getcomposer.org/download/
[11] https://gist.github.com/geschke/d916cf05c976abfc3331387b1c77d624
[12] https://docs.aws.amazon.com/de_de/lambda/latest/dg/nodejs-prog-model-context.html
[13] https://docs.aws.amazon.com/de_de/lambda/latest/dg/intro-invocation-modes.html
[14] https://bref.sh/docs/runtimes/
[15] https://serverless.com/framework/docs/providers/aws/guide/serverless.yml
[16] https://runtimes.bref.sh
[17] https://aws.amazon.com/de/api-gateway/
[18] https://bref.sh/docs/local-development.html
[19] https://github.com/brefphp/bref/tree/master/runtime/php
[20] https://bref.sh/docs/environment/php.html
[21] https://boerse.ard.de/anlagestrategie/branchen/europa-visier-cloud-amazon-microsoft100.html
[22] https://docs.microsoft.com/de-de/azure/azure-functions/functions-overview
[23] https://docs.microsoft.com/de-de/azure/azure-functions/supported-languages
[24] https://www.peachpie.io
[25] https://cloud.google.com/functions/docs/concepts/exec
[26] https://www.alibabacloud.com/help/doc-detail/89029.htm
[27] https://reactphp.org
[28] https://www.kuerbis.org/2019/01/cloud-geschichten-teil-6-serverless-computing-in-der-alibaba-cloud-mit-php/
[29] https://gnugat.github.io/2016/04/13/super-speed-sf-react-php.html
[30] https://github.com/geschke/aws-lambda-php-bref-mailfunc-example
[31] https://bref.sh/docs/#maturity-matrix