Interactive Application Deployment Planning for Heterogeneous Computing Continuums

Daniel Hass and Josef Spillner

Abstract Distributed applications in industry are modular compositions involving containers, functions and other executable units. To make them deployable and executable in production, they need to be assigned to heterogeneous resources in computing continuums, consisting of multiple clouds, devices and other runtime platforms. Existing assignment processes are neither transparent nor interactive. To overcome this limitation, we introduce the Continuum Deployer, a tool capable of reading application descriptions (e.g. Helm charts for Kubernetes), interactively performing and comparing assignment algorithms (e.g. to multiple K8s and K3s cloud/edge/fog/device resources), and exporting deployment files (e.g. Kubernetes manifests). We evaluate the tool with empirical package analysis, exemplary deployments and a synthetic experiment. to appropriately address scalability concerns.

1 Introduction and Problem Statement

Computing continuums combine multi-cloud resources with local devices, including resource-constrained (mobile) edges and fogs [1]. They permit the optimal distribution of complex distributed application functionality along the data path. Sensor data processing continuums, for instance, may perform filtering early at the point of sensing, and complex analytics later in appropriate clouds [2]. Building distributed applications for such infrastructures has become feasible due to the proliferation of many microservice and serverless

Daniel Hass
Endress+Hauser InfoServe, Weil am Rhein, Germany, e-mail: daniel.hass@endress.com

Josef Spillner
Zurich University of Applied Sciences, Distributed Application Computing Paradigms (blog.zhaw.ch/splab/), Winterthur, Switzerland, e-mail: josef.spillner@zhaw.ch

technologies [3] and corresponding packaging formats for compositions and workflows. However, which application part should go to which resource along the continuum is still an open challenge especially in automation-focused industrial domains. We refer to it as *application–continuum resource assignment problem*. Recent research has led to many automated algorithms to solve the assignment problem under constraints and preferences. Proposals encompass the definition of cost/utility functions for the Hungarian algorithm, rule-based matchmaking [4], topology splitting and matching [5], and device-driven adaptive deployment [6]. However, none of these approaches keep the person responsible for deployment in the loop. Many DevOps engineers and administrators would prefer a guided approach where automation is built in but certain decisions can be controlled and followed in a transparent way. Such an anticipated approach is in line with recent trends towards responsible and explainable artificial intelligence, equally applying to automation decisions in upcoming continuums such as decentralised and osmotic computing, cloud robotics, cloud-based manufacturing processes and vehicular clouds [7]. To accommodate this need, this paper introduces the *Continuum Deployer*, a novel interactive tool to solve the assignment problem especially for fast-paced industrial DevOps processes.

2 Solution Space and Method

The solution space to constrained assignment is potentially large in practice due to the following factors:

1. Lack of resource descriptions. Although there are standards like the Common Information Model to describe infrastructure capabilities, they are rarely used in practice. Resources themselves (e.g. mobile phones, smart watches, IoT devices or servers) also do not ship with self-descriptive capabilities.
2. Application description variety. In contrast to resources, engineers have various approaches available to describe their applications and how to deploy them. Modern distributed applications are either deployed through infrastructure as code or through declarative formats, including low-level Docker Compose files and Kubernetes manifests, high-level Helm charts and Cloud-Native Application Bundles (CNAB), Open Application Model (OAM) or Topology and Orchestration Specification for Cloud Applications (TOSCA) files for container-based compositions, and AWS Serverless Application Models (SAM) for function-based compositions [5].
3. Scalability. Each application part may be scaled independently from the others, leading to the need to reserve a multiple of its resource requirements on the same or on different resources.
4. Constraints and preferences. Application descriptions may ship with a-priori constraints on where deployments is allowed. In practice, this infor-

mation is often absent and may have to be added in an interactive and incremental way, in each iteration followed by a check of the resulting deployment plan. The same process is needed for custom preferences – for instance, to declare that although a database could run on the (mobile) edge, it should instead run in the cloud because a later upgrade of the application needs to access it there.

To address this large solution space, Fig. 1 contains the approach that gradually achieves solutions of different maturity (design, implementation, evaluation) for a subset of industrially relevant cloud application technologies. The remaining sections cover the steps of the proposed method and lead to the presentation of an applied deployment tool which we make available as open source software.

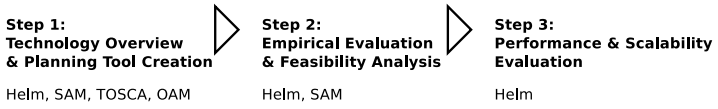

Fig. 1 Solution Approach

3 Tool Design

We assume that both resources and application descriptions have matching properties, in particular: CPU cores, memory capacity, and preference labels such as 'private cloud' or 'in-vehicle', and that these properties are accurately described for at least the majority of these entities. Labels are the central mean within the Continuum Deployer for the user to express certain constraints and preferences with regard to the deployment placement. Each node and workload can be assigned with zero to as many labels as the user desires. A suitable node must or should (depending on configuration) possess all of the workloads labels or more to be considered for a deployment. Unlabelled workloads are able to run on any of the available nodes.

The labels are then used to distinguish resource capabilities, ranging from vast and elastically scalable cloud resources to constrained device resources which, due to these constraints, might not always allow for executing all workloads resulting from the heterogeneous packaging formats. Fig. 2 shows a subset of the problem space, leading to the research question: How can a complex microservice-based application, packaged as Helm chart, AWS SAM, OAM appfile or any other format, be deployed to resources with different resource constraints while adhering to user preferences?

The anticipated tool design thus needs to fulfil the following requirements:

1. Extensible support for multiple application packaging formats, deployment languages and assignment resolver algorithms (matchers).

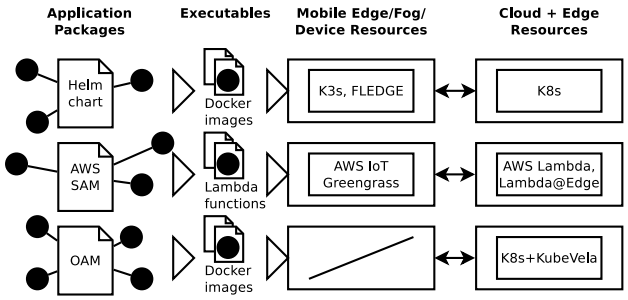

Fig. 2 Schema of deployment planning for various application packaging formats

2. Consideration of scaling factors and the resulting resource multiplicity.
3. Specification and consideration of user-specific preferences, primarily in the sense of preferring one resource over another (i.e. cloud-first, device-first) but also allowing for more fine-grained trade-offs.
4. Interactive as well as non-interactive use, both with tolerance for incomplete resource specifications.

4 Assignment Algorithms

To ensure the functionality of extensible algorithms, we implemented and integrated two exemplary matchers with vastly different capabilities.

1. Greedy search. The greedy solver asks for a single optimisation target. Implemented are the resource and workload sorting and subsequent greedy matching for CPU and memory, although other metrics can be trivially added. The largest workloads are probed for placement on a sorted list of resources. In this list resources appear in descending order based on the selected optimisation target.
2. SAT search. The constraint satisfaction problem (CP-SAT) solver [8] offers multiple options with regard to the optimisation target. Implemented are six single and multiple targets, including the maximisation of idle CPU, the minimisation of idle memory, or the maximisation of idle combined resources. The implementation documentation gives more details on what is implemented and how to implement further combinations. This solver uses constrained programming to define rules and constrains that describe the resource matching problem in mathematical terms. Afterwards this optimisation is solved as optimal as possible. The results of this solver differ from the greedy ones: if this solver cannot come up with an optimal solution the run will fail and all resources are displayed as unschedulable. This feasibility constraint is enforced on each label group (if labels are defined).

5 Tool Implementation

Continuum Deployer is implemented as command-line tool with several input assistance features including syntax completion and adaptive menus. For the purpose of automation, arguments can be passed to reduce or even avoid any interactive step. A state machine is used to control the internal workflow and any shortcuts through passed arguments, as shown in Fig. 3.

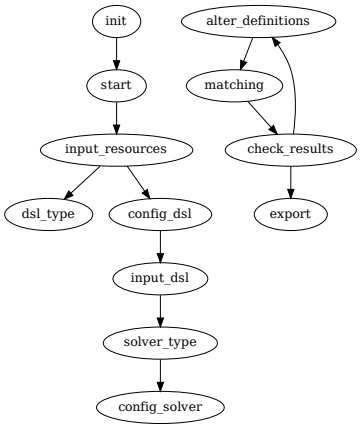

Fig. 3 State machine representing the process of deployment planning

The initial interaction screen implies the parsing of the specified resource definition file and shows the initial resource utilisation. Fig. 4 shows the case of three initially 'empty' resources, referring to all resource metrics (processor utilisation, memory allocation) beyond the necessary minimum occupied by the operating system, virtualisation environment and cloud stack (e.g. Kubernetes).

Afterwards, the user selects a domain-specific language, and furthermore selects among given file format options (Fig. 5). Eventually, an application description in the selected domain-specific language (DSL) and format is loaded, and the matchmaking to assign the set of application parts to the set of resources begins. The deployer implementation supports Helm as DSL with local, templated and packaged Helm charts as flavours.

The matchmaking performs the user-selected algorithm (SAT, greedy or user-extensible search). The SAT solver is implemented with the Google OR tools for combinatorial optimisation.

Eventually, the best assignment is visualised. Even the best one may not be sufficient and requires a (minimum) amount of additional resources. In the given scenario (Fig. 6), a basic deployment of the Nginx web server is possible but a replicated deployment is not, as evidenced by the inability to find a suitable spot for the replica `nginx-deployment-1-2`.

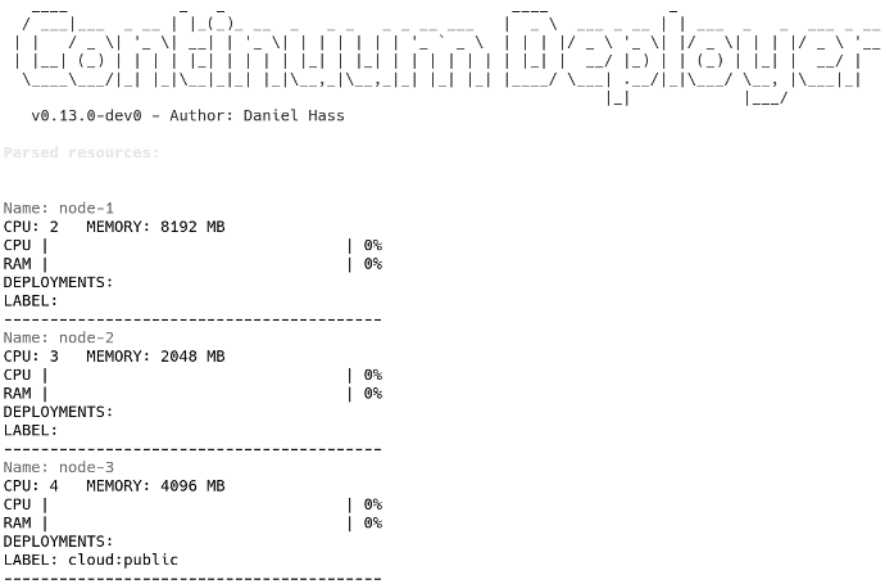

Fig. 4 Start screen of Continuum Deployer after resource parsing

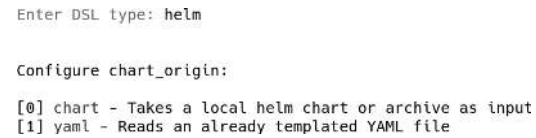

Fig. 5 Selection of cloud application DSL and packaging format

The chosen assignment can be refined by interactively tuning the resource definitions, for instance removing a resource that turned out to be underutilised. Eventually, the resulting deployment plan can be exported into the Kubernetes manifest format. This allows the use of a separate deployment tool, ranging from simple `kubectl` invocations in DevOps scenarios to sophisticated GitOps deployers [9], to turn the plan into an actual deployment.

6 Evaluation

6.1 Empirical application evaluation and feasibility analysis

We evaluate two popular and industrially relevant packaging formats, Helm charts for container compositions and SAM for serverless applications. We are interested in knowing how many of them specify resource constraints in

```
Matching results:


Name: node-1
CPU: 2   MEMORY: 8192 MB
CPU |                                  | 0%
RAM |                                  | 0%
DEPLOYMENTS:
LABEL:
----------------------------------------
Name: node-2
CPU: 3   MEMORY: 2048 MB
CPU |####################              | 66%
RAM |########                          | 25%
DEPLOYMENTS:
         nginx-deployment-4, cpu=2.0, memory=512, label=[]
LABEL:
----------------------------------------
Name: node-3
CPU: 4   MEMORY: 4096 MB
CPU |########################          | 80%
RAM |###################               | 62%
DEPLOYMENTS:
         nginx-deployment-1-0, cpu=0.4, memory=512, label=[cloud:public]
         nginx-deployment-1-1, cpu=0.4, memory=512, label=[cloud:public]
         nginx-deployment-1-2, cpu=0.4, memory=512, label=[cloud:public]
         nginx-deployment-3, cpu=2.0, memory=1024, label=[]
LABEL: cloud:public
----------------------------------------

[Error] The following workloads could not be scheduled:
Name: nginx-deployment-2
CPU: 4.0         MEMORY: 512 MB
LABEL:
----------------------------------------
```

Fig. 6 Best (albeit invalid) greedy match between application and resources

order to make them eligible for a resource-aware distributed multi-cloud or edge-cloud deployment. All evaluations are conducted with October 8, 2020 data snapshots.

Due to the popularity of Helm, several public marketplaces to share Helm-packaged cloud applications exist, such as KubeApps Hub, Helm Hub and Artifact Hub. We include 306 Helm charts retrieved from KubeApps Hub into our analysis. They reference 459 deployable container images. Out of those, 114 are resource-constrained, with 113 constraining memory and 109 constraining CPU cycles. The physical units differ - 93% use `Mi` to refer to memory in mebibytes (multiple of 2^{20} bytes), while the remainder uses `Gi` as well `M` (multiple of 10^6 bytes). For CPU usage, 97% use `m` to refer to milli-vCPU, while the remainder uses fractions of vCPUs (`1`, `0.1`). To facilitate the comparison, we unify all units to mebibytes and vCPU fractions, respectively. Fig. 7a/b contain the resulting breakdowns of memory and CPU constraints.

Moreover, we compare with 535 SAM files retrieved from the AWS Serverless Application Repository. With 63 of them bundling multiple deployable Lambda functions, they reference a total of 615 Lambda functions. Out of those, 387 are resource-constrained; due to the Lambda execution model that allocates CPU cycles proportionally to memory, only the memory size is specified explicitly. In contrast to helm, the physical units are equalised, and allocations are restricted to a small set of possible values. Fig. 8 contains

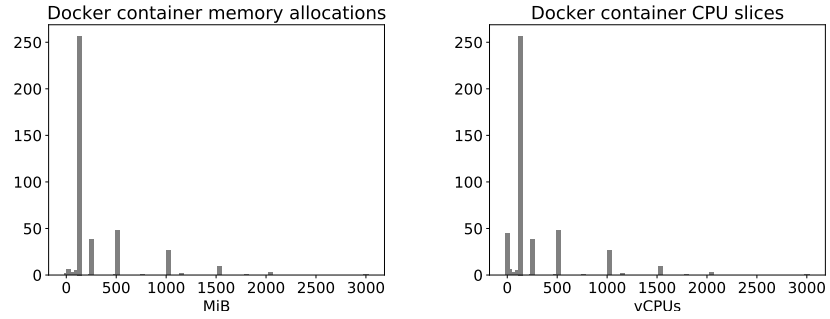

Fig. 7 Frequency of resource allocations for Docker containers in Helm charts

the breakdown of memory allocations, clearly showing a dominance of small functions that can run on resource-constrained devices.

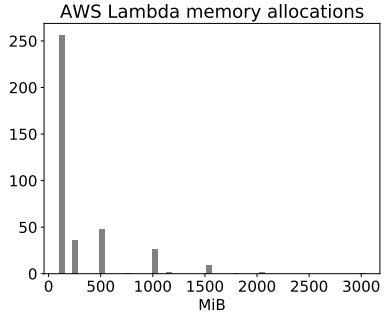

Fig. 8 Frequency of resource allocations for Lambda functions in SAM files

The analysis of Helm charts and SAM files shows that our assumption that at least a significant portion of applications packages ship with maintained resource constraints holds true and therefore the deployer is able to perform useful work.

6.2 Exemplary performance evaluation

We measure the placement determination of two popular Helm charts, `wordpress` and `redis`, with the two built-in algorithms, greedy solver and SAT solver, in various configurations to convey the practical feasibility of incorporating a placement decision in real-time as part of deployment workflows. Both charts are modest in size and are fed to the solver in different formats. Redis consists of one secret, two config maps, three services and

two stateful sets, or a total of eight Kubernetes objects in rendered template format. Wordpress consists of a dependency chart, `mariadb`, and seven Kubernetes objects. in compressed chart format.

All measurements are averaged across 100 invocations to reduce the influence of outliers. Fig. 9 summarises the results. There are no grave differences between any of the algorithm combinations, although evidently the Redis numbers are lower due to the already rendered charts. Moreover, the performance is worst for the holistic idle resource maximisation for Wordpress, whereas it is worst for idle CPU-only maximisation for Redis. In all cases, the DevOps overhead is less than half a second and thus acceptable to trigger the matching from code commits.

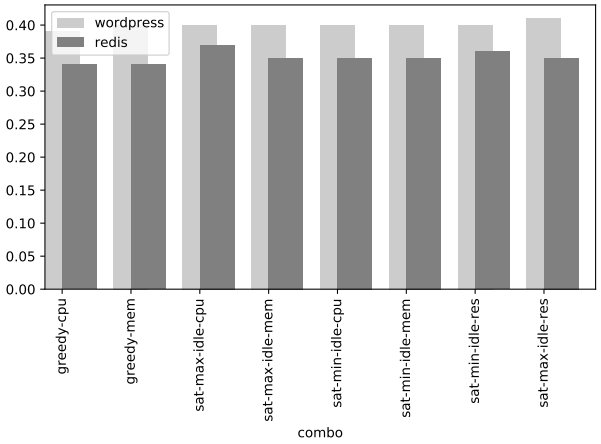

Fig. 9 Performance of placement determination algorithms

6.3 Synthetic scalability evaluation

The assignment problem is NP-hard and thus requires specific scalability techniques for interactive matching to ensure that any chosen algorithm will remain controllable by the user. We assign up to 10 synthetically generated multi-container applications to 10 randomly determined resources. While this typically yields a first feasible solution within one second through the SAT solver configured for maximum idle resource optimisation, finding the optimal solution even for only 4 applications may take up almost 3 minutes on a 2.6 GHz CPU. The Continuum Deployer addresses this problem by (i) showing intermediate results, (ii) letting the user accept those results instead of waiting for the optimum, and (iii) allowing for roundtripping with ad-

justed configuration and relaxed constraints. The batch mode thus allows for inclusion in latency-critical processes such as CI/CD hooks assuming that non-optimal deployments are acceptable.

7 Conclusions and Material

With this paper, we have introduced the Continuum Deployer as solution for transparent and interactive deployments of composite applications to heterogeneous resources across computing continuums. We followed a systematic approach to contribute a practically useful and extensible tool and to evaluate some of the possible combinations between application packaging formats and solution algorithms. The implementation of Continuum Deployer is available as open source at `https://doi.org/10.5281/zenodo.4584220`.

References

1. D. Balouek-Thomert, E.G. Renart, A.R. Zamani, A. Simonet, M. Parashar, Int. J. High Perform. Comput. Appl. **33**(6) (2019). DOI 10.1177/1094342019877383
2. M.D. Donno, K. Tange, N. Dragoni, IEEE Access **7**, 150936 (2019). DOI 10.1109/ACCESS.2019.2947652
3. J.A. Añel, D.P. Montes, J.R. Iglesias, *Cloud and Serverless Computing for Scientists - A Primer* (Springer, 2020). DOI 10.1007/978-3-030-41784-0
4. J. Spillner, P. Gkikopoulos, A. Buzachis, M. Villari, in *2nd International Workshop on Cloud, IoT and Fog Systems/Security (CIFS) / 14th IEEE/ACM International Conference on Utility and Cloud Computing (UCC)* (2020)
5. K. Saatkamp, U. Breitenbücher, O. Kopp, F. Leymann, in *CLOSER 2017 - Proceedings of the 7th International Conference on Cloud Computing and Services Science, Porto, Portugal, April 24-26, 2017*, ed. by D. Ferguson, V.M. Muñoz, J.S. Cardoso, M. Helfert, C. Pahl (SciTePress, 2017), pp. 247–258. DOI 10.5220/0006371002470258
6. T. Quang, Y. Peng, in *2020 IEEE International Conference on Pervasive Computing and Communications Workshops, PerCom Workshops 2020, Austin, TX, USA, March 23-27, 2020* (IEEE, 2020), pp. 1–6. DOI 10.1109/PerComWorkshops48775.2020.9156140
7. P. Wintersberger, H. Nicklas, T. Martlbauer, S. Hammer, A. Riener, in *AutomotiveUI '20: 12th International Conference on Automotive User Interfaces and Interactive Vehicular Applications, Virtual Event, Washington, DC, USA, September 21-22, 2020* (ACM, 2020), pp. 252–261. DOI 10.1145/3409120.3410659
8. G.D. Col, E. Teppan, in *Proceedings 35th International Conference on Logic Programming (Technical Communications), ICLP 2019 Technical Communications, Las Cruces, NM, USA, September 20-25, 2019*, *EPTCS*, vol. 306, ed. by B. Bogaerts, E. Erdem, P. Fodor, A. Formisano, G. Ianni, D. Inclezan, G. Vidal, A. Villanueva, M.D. Vos, F. Yang (2019), *EPTCS*, vol. 306, pp. 259–265. DOI 10.4204/EPTCS.306.30
9. J. Spillner, D. Boruta, T. Brunner, S. Gerber, A. Kosmaczewski. Syn: GitOps on Stereoids with Kubernetes the Swiss Way. 3rd International Conference on Microservices (Microservices) (2020)