DOI: 10.22032/dbt.57789

Simulation in Produktion und Logistik 2023
Bergmann, Feldkamp, Souren und Straßburger (Hrsg.)
Universitätsverlag Ilmenau, Ilmenau 2023
DOI (Tagungsband): 10.22032/dbt.57476

On the Usage of Container and Container Orchestrators as a Computational Infrastructure for Simulation Experiments

Containerisierung und Container Orchestrierung als high-performance Infrastruktur für Simulationsexperimente

Daniel Seufferth, Heiderose Stein, Falk Stefan Pappert, Oliver Rose, Universität der Bundeswehr München, Neubiberg (Germany) daniel.seufferth@unibw.de, heiderose.stein@unibw.de, falk.pappert@unibw.de, oliver.rose@unibw.de

Abstract: The execution of simulation experiments becomes increasingly resource-intensive, either due to the increasing scale and complexity of the simulation model or the nature of the experiment itself. Popular approaches, like simulation-based optimisation and data farming, require extensive computational resources. Therefore, an appropriate methodology is needed to distribute simulation workloads onto complex computing infrastructures efficiently. Containerisation and container orchestration are promising approaches toward this goal. This paper discusses the requirements needed to utilise containerisation for simulation, gives an example of a container-based computational infrastructure for experimentation, and shows how containerisation and container orchestration benefit simulation execution.

1 Introduction

Computational power is a limiting factor for simulation. Over the past decades, the increasing performance of personal computer hardware allowed focusing the execution of simulation experiments on the use of single PCs. When the complexity of one single simulation run – or the sheer number of simulation experiments as in data farming or simulation optimisation (Lechler et al. 2021) – exceeds the capacity of a single PC (see Sanchez et al. 2021), there is a need to use more powerful computing systems (Król et al. 2013) as an environment for running simulations.

Evaluation of how simulation experiments can be sped up using high-performance computing infrastructures began in the 1970s (Taylor 2019). The ongoing trend of cloud computing and virtualisation techniques offers an interesting opportunity to revisit this topic: containerisation can encapsulate the simulation and makes it independent of the cluster technology. And by using container orchestration tools, scalability becomes readily available for simulation experimentation.

The research on high-performance computing infrastructures is diverse and growing. For example, Scalarm and DIRAC present different computing infrastructures to distribute simulation workloads onto multiple computing nodes in heterogeneous environments (Król et al. 2013). Anagnostou et al. 2019 evaluated recent technological approaches with their work on simulation experimentation frameworks, applying a micro-services auto-scaling approach utilising MiCADO. MiCADO focuses on efficiently utilising cloud resources, dynamically scaling the Kubernetes cluster it uses for container orchestration. It also extends the manifest files for Kubernetes Application Programming Interface (API) objects with its Application Description Templates (MiCADO Project 2023). This makes MiCADO less universal and adds the risk of delayed implementation to changes in Kubernetes API objects and extensions of upstream Kubernetes. Reviewing the recent literature, to the authors' knowledge, no solution is readily available to containerise a given simulation tool and run it on a cluster.

We currently do not see a significant uptick in adopting these methods within the simulation community, neither on the side of the practitioner nor the side of software vendors. We generally observe a need for more awareness in our community, which we would like to address by first discussing the requirements to use these technologies for simulation purposes. Additionally, we show a straightforward approach to building a container-based computer cluster as a framework for running simulation experiments to make this technology more accessible.

The remainder of this article is structured as follows: First, we discuss some requirements and prerequisites for containerisation and container orchestration of simulation workloads. This is followed by describing how we set up the simulation cluster at the University of the Bundeswehr Munich. To illustrate the capabilities of different computing infrastructures, we give an overview of experimentation results and how Kubernetes benefits the user in addressing more computational resources for simulation workloads. Lastly, as an outlook, we propose possible next steps and research opportunities.

2 Requirements to use containerisation for simulation

Using containers for simulation execution comes with different requirements for the simulation engines. We group these into three categories, which will be discussed in the following subsections:

- Modelling requirements
  - Automated model generation
- Integration
  - External trigger
  - Headless simulation
- Usability in container environments
  - Operating system
  - Ephemeral simulation models
  - Licensing
  - Containerization

2.1 Modelling requirements

Modelling requirements describe how well models can be exchanged with a simulation package. The simulation package needs to support the creation of models by external systems. Especially with running simulation experiments on a larger scale, where it is more feasible to generate new models instead of parameterising a hand-made model, the ability to automatedly generate models for a given simulator becomes increasingly important. Simulation packages allow automated ways of model building to different degrees. Higher levels mean more of the generator is independent of the simulation package, allowing for more flexibility and possible reuse for other packages. Several levels of automation can be considered:

1. No automation
2. Parameterization of a hand-made model
3. Bootstrapping models based on external data
4. External generation of model files
5. Online model generation using an API

Although automated model generation is not a new topic, its increasing relevance in the face of online simulation and digital twins has yet to lead vendors to support higher-level automation approaches. Therefore, careful consideration of automated model generation capabilities should be given when scaling up model execution.

2.2 Integration

Integration requirements describe how well a simulation package can be integrated into another software system, e.g., how well it can be started and executed. For execution, generally, containerised software in Kubernetes is running headless. Graphical User Interfaces (GUIs) allowing access to the backend software may be run as independent web services if needed. This fundamental aspect of containers affects the usage of containerisation technology for simulation execution.

Besides running headless, how a simulation run can be triggered is a big concern. In most simulation environments, the user starts a simulation run by clicking a button on the GUI. Automating simulation runs, this is no longer a feasible solution. Although most simulation packages bring their simulation execution environment, these are typically not yet designed to support simulations on distributed infrastructures. Building such systems requires developing an experiment manager and an external way to trigger a simulation run. The most common way this is supported is by using command-line interfaces. A more convenient option could be starting simulations utilising an API.

2.3 Usability in container environments

2.3.1 Operating system

Containerisation is a form of operating system virtualisation. (Huawei Technologies Co. Ltd. 2023) summarises the evolution of containerisation, beginning with the chroot-command and finishing with modern containerisation formats like Docker or LXC. As containers use the kernel of the host operating system (OS), Linux-based containers can only run on hosts with Linux. This, in turn, means that the simulation model to be containerised must run on Linux, restricting the number of simulation engines that can be used for model creation as only some engines support Linux.

Simulation engines requiring Windows can only be run as a Windows container. This means that a hybrid cluster, consisting of both Linux and Windows worker nodes, is necessary, as the control plane of Kubernetes must run on Linux (Kubernetes 2023). A hybrid configuration of Kubernetes generates a performance loss, as it is no longer possible to allocate all the available resources when nodes of several operating systems need to be provided. This mix of systems increases the complexity of resource allocation.

Table 1: List of simulation tools and the support of Linux-based OSs

Simulation tool	Linux support?
ANSYS	Yes
Anylogic	Yes
Factory Explorer	No (Windows only)
Flexsim	No (Windows only)
Matlab/Simulink	Yes
OpenModelica	Yes
Simio	No (Windows only)

Table 1 gives an overview of different simulation tools we have used for various purposes (discrete event simulation, fluid simulation, etc.) and whether they support Linux. This small overview shows that although Linux support is not an exception, even popular simulation tools require Windows.

2.3.2 Ephemeral simulation models

As a container orchestrator, Kubernetes aims to have all containers running. If containers fail, it automatically restarts them as new instances, possibly on a different worker node. This characteristic of containers is called ephemeral, meaning they can get killed and restarted at any given time due to internal or external causes, e.g., software updates, load balancing, or simulation failures. Therefore, simulation models that run as containers in Kubernetes should support this ephemeral nature. Here are a few ideas on how to handle simulation in such an environment:

- One of the easiest ways of handling the ephemeral nature of containers inside Kubernetes is accepting the killing of execution runs and restarting it. This comes with the question of monitoring lost simulation runs. Kubernetes has several extensions that provide extensible monitoring and logging features, which can be used to keep track of prematurely stopped scenario calculations.
- Keeping track and storing the progress of simulation runs. Therefore, killed instances of the model can be restarted without losses using cached data. To keep an up-to-date image of the current simulation state in storage, continuous updates to this image are required, which causes a significant amount of data to be transferred and stored. Besides the general cost of storing, there will likely be an adverse effect on simulation speed due to these tasks. Therefore, this approach is only feasible in an environment where successfully finishing simulation runs is otherwise unlikely due to very long simulation runs or unstable computing hardware.

- Starting additional instances, adding redundancy, and decreasing the risk of data loss in case of a container failure. As most simulations already utilise replications to achieve sufficient confidence intervals, adding additional runs depending on stability would not increase the load by a large margin.

Using containerisation technologies for simulating models at scale brings with it the need to consider the premature termination of simulation containers. This affects the number of simulation runs that can be completed. Furthermore, it requires consideration of the data collected during one run. While simulation models that purely run in memory usually leave no trace when their surrounding container disappears, simulation models reporting data during the simulation run to a database require more involved handling as incomplete data needs to be cleaned in case the container fails.

2.3.3 Licensing problems

The software vendor's approach to licensing is also a topic worth considering when deciding on a simulation package. When considering licensing, the cost structure should be reviewed. There are simulation packages where costs depend only on the model development environment. This allows very flexible scaling of the simulation to new or different projects. Then there are per-user/seat licenses, which can be challenging to scale in a legally safe way. A third option is a licensing model, which charges per core, which can get expensive when sufficient hardware is available.

Another side to licensing, besides the cost and legality, is the way the developer enforces licensing. From our point of view, licensing servers with concurrent licensing are ideal for a dynamic environment like Kubernetes. Any licensing scheme enforced by hardware restrictions significantly limits its use in a container environment. Examples are license codes tailored to a specific PC or hardware keys provided as USB dongles.

2.3.4 Containerization

Creating container images depends on the choice of the container engine. Multiple engines like Docker (Docker Inc. 2023), Podman (Podman 2023) or Apptainer (Apptainer Project 2023) are available, all of which have a similar approach to image creation, utilising a descriptive file that defines all steps necessary for containerisation, which we will call the "containerisation file" in the following.

The containerisation file generally begins with a base image, e.g., an Alpine Linux image, and defines which libraries and binaries to add. In the case of a simulation model, you need to package your simulation tool of choice so the simulation engine is available inside the container. Supporting installation via the command line, containerisation of the chosen simulation tools is done by adding a single command to the containerisation file. If this simple installation method via the command line is not supported, a more tedious approach must be utilised for installing the simulation tool.

A wholly optimised containerisation process for simulation models means that simulation package developers provide usable images of their simulation software, streamlining the containerisation process of simulation models.

In a perfect world, simulation software developers would provide a base image of their simulation package, where the user only needs to add their model.

3 Our current setup

Creating our simulation cluster based on Kubernetes has been an iterative process, starting with small test configurations and moving to clusters sufficient for large-scale parallelised simulation experiments. This section describes the setup of the simulation cluster we currently use at the University of the Bundeswehr Munich.

The cluster runs on a server that consists of 40 hosts. Instead of running Kubernetes on bare-metal – meaning that the cluster has access to the OS of the hosts – a virtualisation layer is between the physical hardware and the Kubernetes cluster. The usage of virtualisation has two benefits:

1. Added flexibility: instead of purely running Kubernetes on the cluster, virtualisation enables us to run independent virtual machines for particular use cases, e.g., different guest operating systems.
2. Added security: the virtualisation layer isolates the host kernel from the kernels of the guests used by Kubernetes.

Rancher – a cluster management tool – is used to configure the cluster. Rancher itself runs as a container inside of a smaller Kubernetes cluster. Ranchers WebUI allows the configuration of the virtual machines that run Kubernetes, automatically creates the virtual machines, and sets up the Kubernetes cluster. Rancher uses a particular distribution of Kubernetes called RKE2 (Rancher Kubernetes Engine 2), which is more lightweight than upstream Kubernetes and focuses on security (Rancher Labs 2023). It also comes with “containerd” (containerd 2023) as the embedded container runtime and supports hybrid cluster configurations consisting of both Linux and Windows worker nodes. Rancher not only automates the cluster creation process but also simplifies the installation of additional software, e.g., monitoring software or storage provisioning, through integrated software repositories and automated installation utilising Helm charts.

In line with best practices, the current simulation cluster of the University of the Bundeswehr Munich consists of multiple control planes and worker nodes. Each of them is running on Ubuntu. The control plane is composed of three nodes, ensuring high availability, that do not run simulation workloads, therefore needing far fewer resources. Six cores and six gigabytes of Random-access Memory (RAM) are sufficient for the workloads the control plane nodes have to execute. We currently use a basic set of four worker nodes, which provide the infrastructure to compute simulation workloads. The virtual machines are configured to consume almost all host resources, using 60 of the 64 cores available and one terabyte of RAM. This provides a computational power of 240 cores and four terabytes of RAM for this cluster. This represents about 10 % of the resources available. If more computing power is required, Rancher allows us to quickly scale the cluster up, adding worker nodes and increasing the cluster’s computational resources. Figure 1 visualises the simulation cluster in detail.

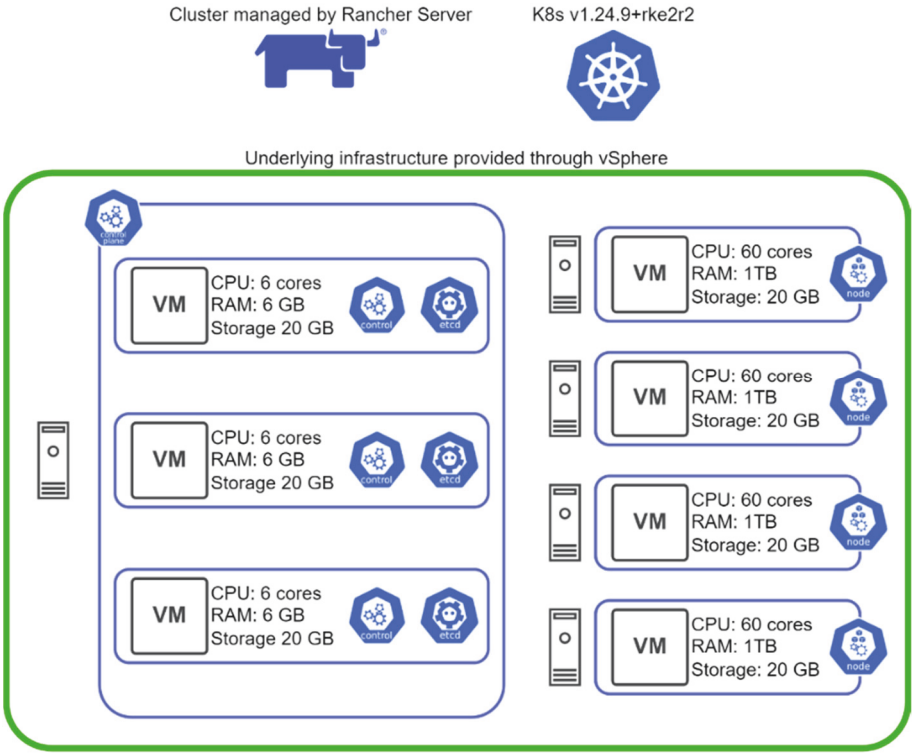

Figure 1: Setup of the current simulation cluster

4 Performance testing

The first test case run on our simulation cluster is based on our work on utilisation thresholds for equipment groups described in (Pappert et al. 2017). The simulation engine used in this project is an in-house development in Java, with the model being created by simply calling methods of the meta-model. With Java being platform-independent, containerisation was a simple process, only needing a base container image that provides the Java Runtime Environment. Furthermore, the experiment engine was designed with distributed execution in mind, meeting many of the requirements for utilising containerisation mentioned above.

With our historical changes to the computational infrastructure used, we can retrace the performance gain achieved by utilising different setups, finally leading to containerisation and Kubernetes. Figure 2 visualises the performance – measured by the number of scenarios evaluated per day – based on other computational infrastructures. The shown infrastructures use different underlying hardware; therefore, the depicted performance gain can primarily be attributed to the extension of the hardware resources. The last two entries show our current simulation server in two different levels of scaling. The primary benefit we see in using Kubernetes comes with its ease of scaling. Once a small cluster is set up on a small portion of hardware, scaling available processing power to full capacity is but a matter of a few minutes.

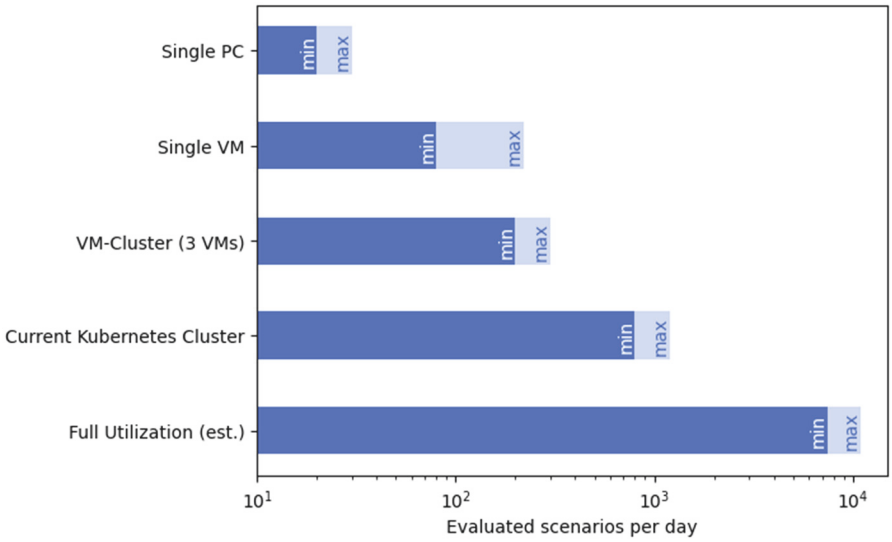

Figure 2: Performance comparison based on different computational infrastructures shows the number of scenarios calculated daily for each setup

5 Summary and further research

We have shown an example setup using Rancher and Kubernetes to facilitate the creation of large-scale computational infrastructure. It is built solely with free and open-source software so that it can be used as a template for other Kubernetes clusters. As the software used supports several kinds of infrastructures – different cloud providers and on-premises infrastructure – it is flexible and can be used in environments that differ from the one discussed here. The resulting cluster simplifies the allocation of computing resources, making more extensive computing infrastructures accessible to the user.

We have discussed different requirements regarding utilising containerisation for simulation workloads on distributed infrastructures. Several of these points should be further researched to streamline the containerisation process of simulation models and increase the acceptance of containerisation in the simulation community.

Acknowledgements

We want to thank Uwe Langer and Alexandros Karagkasidis for their continuous hardware infrastructure support.

This research is funded by dtec.bw – Center of Digitalization and Technology Research of the Bundeswehr.

dtec.bw is funded by the European Union – NextGenerationEU.

References

Anagnostou, A.; Taylor, S.J.; Abubakar, N.T.; Kiss, T.; DesLauriers, J.; Gesmier, G.; Terstyanszky, G.; Kacsuk, P.; Kovacs, J.: Towards a deadline-based simulation experimentation framework using micro-services auto-scaling approach. In: Mustafee, N.; Bae, K.-H.G.; Lazarova-Molnar, S.; Rabe, M.; Szabo, C.; Haas, P. and Son, Y.-J. (Eds.): Proceedings of the 2019 Winter Simulation Conference (WSC), National Harbor (USA), December 8th-11th December 2019, pp. 2749–2758.

Apptainer Project, 2023: Documentation | Apptainer. https://apptainer.org/docs/, accessed May 11th, 2023..

containerd, 2023: containerd overview. https://containerd.io/docs/, accessed May 12th, 2023.

Docker Inc., 2023: Docker Engine overview. https://docs.docker.com/engine/, accessed May 11th, 2023.

Huawei Technologies Co., Ltd.: Container technology. In: Cloud computing technology: Springer, Singapore 2023, pp. 295–342.

Król, D.; Wrzeszcz, M.; Kryza, B.; Dutka, Ł.; Kitowski, J.: Massively scalable platform for data farming supporting heterogeneous infrastructure. In: Zimmermann, W. (Eds.): The 4th International Conference on Cloud Computing, Grids, and Virtualization, Valencia (Spain), May 27th-1st June, pp. 144–149.

Kubernetes, 2023: Windows containers in Kubernetes. https://kubernetes.io/docs/concepts/windows/intro/, accessed March 28th, 2023.

Lechler, T.; Sjarov, M.; Franke, J.: Data Farming in Production Systems - A Review on Potentials, Challenges and Exemplary Applications. Procedia CIRP 96 (2021), pp. 230–235.

MiCADO Project, 2023: Application Description Template - MiCADO. https://micado-scale.github.io/adt/, accessed April 13th, 2023.

Pappert, F.S.; Rose, O.; Suhrke, F.; Mager, J.: Simulation based approach to calculate utilization limits in opto semiconductor frontends. In: Chan, V.W.K.; D'Ambrogio, A; Zacharewicz, G; Mustafee, N.; Wainer, G. and Page, E.H. (Eds.): Proceedings of the 2017 Winter Simulation Conference (WSC), Las Vegas (USA), December 3rd-6th December 2017, pp. 3888–3898.

Podman, 2023: Getting Started with Podman | Podman. https://podman.io/docs, accessed May 11th, 2023.

Rancher Labs, 2023: Introduction | RKE 2. https://docs.rke2.io/, accessed May 5th, 2023.

Sanchez, S.M.; Sanchez, P.J.; Wan, H.: Work smarter, not harder: A tutorial on designing and conducting simulation experiments. In: Kim, S.; Feng, B.; Smith, K.; Masoud, S.; Zheng, Z.; Szabo, C. and Loper, M. (Eds.): Proceedings of the 2021 Winter Simulation Conference (WSC), Phoenix (USA), December 13th-17th December 2021, without page numbers.

Taylor, S.J.: Distributed simulation: state-of-the-art and potential for operational research. European Journal of Operational Research 273 (2019) 1, pp. 1–19.