Kalpa Publications in Computing
Volume 21, 2025, Pages 45–60
Proceedings of the 14th and 15th International Workshops on the Implementation of Logics

Towards StarExec in the Cloud

David Fuenmayor[1], Jack McKeown[2], and Geoff Sutcliffe[2]

[1] University of Bamberg, Bamberg, Germany
david.fuenmayor@uni-bamberg.de
[2] University of Miami, Miami, USA
jam771@miami.edu,geoff@cs.miami.edu

Abstract

StarExec has been central to much progress in logic solvers over the last 10 years. It was recently announced that StarExec Iowa will be decommissioned, and while StarExec Miami will continue to operate while funding is available, it will not be able to support all the logic solver communities currently using the larger StarExec Iowa. In the long term StarExec will necessarily have to migrate to new compute environments. This paper describes work being done to reengineer StarExec as a cloud-native application using container technology and infrastructure-as-code practices. The first step has been to containerise StarExec and ATP systems so that they can be run on a broad range of computer platforms. The next step in process is to write a new backend in StarExec so that Kubernetes can be used to orchestrate distribution of StarExec job pairs over whatever compute nodes are available. Supported by an Amazon Research Award, a new version of StarExec will be deployed in AWS.

1 Introduction

Automated Theorem Proving (ATP) is concerned with the development and use of tools that automate sound reasoning: the derivation of conclusions that follow inevitably from facts. Automated Theorem Proving (ATP) is at the heart of many computational tasks, in particular for verification [12, 10] and security [8].[1] New and emerging application areas include chemistry [44], biology [6], medicine [14], elections [21, 4], auctions [5], privacy [18], law [24], ethics [9], religion [22, 2, 16], and business [11]. ATP systems are also used as components of more complex Artificial Intelligence (AI) systems, and the impact of ATP is thus extended into many facets of society.

The Thousands of Problems for Theorem Provers (TPTP) World [40] is a well established infrastructure that supports research, development, and deployment of ATP systems. The TPTP World includes the TPTP problem library [39], the TSTP solution library [37], standards for writing ATP problems and reporting ATP solutions [41, 36], tools and services for processing ATP problems and solutions [37], and it supports the annual CADE ATP System Competition (CASC) [38]. Since its first release in 1993 the ATP community has used the TPTP World as an appropriate and convenient infrastructure for ATP system development, evaluation, and

[1] In AWS - aws.amazon.com/what-is/automated-reasoning/, aws.amazon.com/security/provable-security/.

application. The TPTP World has a diverse, engaged, and sustained user community, with various parts of the TPTP World being deployed in a range of applications in both academia and industry.[2] The web page `www.tptp.org` provides access to all components.

The TPTP problem library was motivated by the need to provide support for meaningful ATP system evaluation. The need to provide support for meaningful system evaluation has been recognized in many other logic solver communities, e.g., TPTP [42], SAT [15], SMT [7], Termination [19], etc. For many years testing of logic solvers was done on individual developers' computers. In 2010 a proposal for centralised hardware and software support was submitted to the NSF, and in 2011 a $2.11 million grant[3] was awarded. This grant led to the development and availability of StarExec Iowa [33] in 2012, and a subsequent $1.00 million grant[4] in 2017 expanded StarExec to Miami. StarExec has been central to much progress in logic solvers over the last 10 years, supporting 16 logic solver communities, used for running many annual competitions [1], and supporting many many users. StarExec Iowa provides community infrastructure for many logic solver communities, e.g., ASP, QBF, SAT, SMT, Termination, etc, while StarExec Miami is used by the TPTP community. StarExec Miami has features that take advantage of TPTP standards, and is also used to host CASC.

It was recently announced that StarExec Iowa will be decommissioned. The maintainer of StarExec Iowa explained that "the plan is to operate StarExec as usual for competitions Summer 2024 and Summer 2025, and then put the system into a read-only mode for one year (Summer 2025 to Summer 2026)". The 2017 grant for StarExec Miami paid for the hardware and three years of system administration. The hardware is still hosted by the University of Miami High Performance Computing group, funded on a shoe-string budget by the TPTP World. While StarExec Miami will continue to operate while funding is available, it will not be able to support all the logic solver communities currently using the larger StarExec Iowa. In the long term StarExec will necessarily have to migrate to new compute environments, and several plans are (at the time of writing) being discussed. This paper describes work being done to reengineer StarExec as a cloud-native application using container technology and infrastructure-as-code practices. The first step has been to containerise[5] StarExec and ATP systems so that they can be run on a broad range of computer platforms. The next step in process is to write a new backend in StarExec so that Kubernetes can be used to orchestrate distribution of StarExec job pairs over whatever compute nodes are available. Supported by an Amazon Research Award (see Section 5) a new version of StarExec will be deployed in AWS. This StarExec instance will be fully functional and available to the community (as much as our budget allows). It will also serve as an exemplary implementation for those willing to deploy their own, possibly customized, StarExec on their own computers or in the cloud.

This paper is organized as follows: Section 2 provides a short background to ATP systems, StarExec, and containerisation. Section 3 describes how StarExec has been containerised, and Section 4 describes how ATP systems have been containerised. Section 5 explains how the containerised StarExec and ATP systems will be deployed in a Kubernetes setting. Section 6 concludes and looks forward to future work.

All the software described in the paper is available from . . .
`github.com/StarExecMiami/StarExec-ARC`.

[2]TPTP has contributed to recognized research in 627 publications that cite [39], according to Google Scholar.

[3]NSF Awards 1058748 and 1058925, led by Aaron Stump and Cesare Tinelli at the University of Iowa

[4]NSF Award 1730419

[5]Strictly, "images" are built, and the images are deployed in containers. But keeping with common use of the terminology, we say "container images" and "containerise".

2 Background

2.1 StarExec

Figure 1 shows the architecture of the currently deployed StarExec Miami. The hardware consists of a single head node and multiple compute nodes. The head node provides the browser interface for users, in particular it accepts job requests that generate job pairs consisting of an ATP system and a problem file, does internal scheduling, and uses the SUN Grid Engine (SGE) to distribute the job pairs to the compute nodes. (For development and testing, the head node can also run job pairs itself using a local backend.) The head node maintains a relational MariaDB database, and all the nodes access an NFS mounted shared file system. The database records everything, including locations of the ATP systems' files and the problem files in the file system. Job pairs executing on a compute node have their time and memory usage limited and reported by the `runsolver` [26] utility (the `BenchExec` [3] utility in StarExec Iowa). The results and resource usage data from completed job pairs are stored in the file system, and recorded in the database. The browser interface provides the necessary facilities for user management, uploading ATP systems, uploading problem files, browsing the ATP systems and problems, creating jobs, imposing resource limits in jobs, tracking job progress, browsing and downloading job results, and deleting ATP systems, problems, jobs, etc.

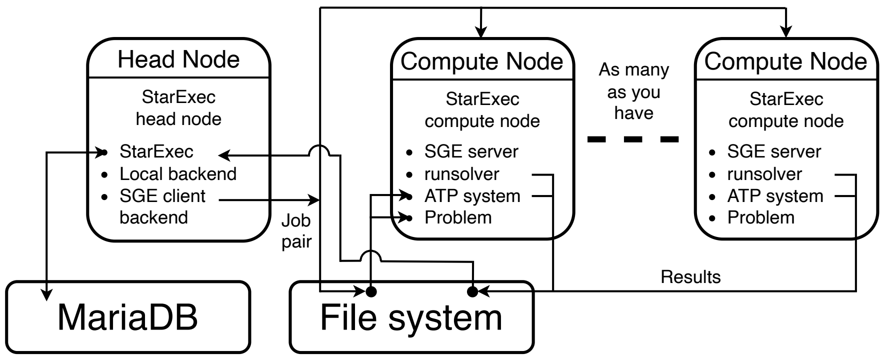

Figure 1: StarExec Architecture

2.2 ATP Systems

ATP Systems are complex pieces of software, typically using advanced data structures [28], sophisticated algorithms [43], and tricky code optimizations [27]. They are written in a variety of programming languages: Prolog [23, 13], Scala [32], C [29], C++ [25], OCaml [17], Python [30], etc. Their build processes include techniques such as parser generators [31], Makefiles, code repositories, specific versions of libraries, etc. For a user who is focussed on an application of ATP, installing an ATP system can be a deal breaker. Many early users selected a weaker system, e.g., Otter [20], for their experiments because it was readily available and easy enough to install. There have been some proposals for standardising the ATP system build process, e.g., `tptp.org/Proposals/SystemBuild.html`, but the diversity of ATP system software makes conformity nigh impossible. An alternative is to push the task back on the system developers, and one approach to this is containerising ATP systems, as discussed in Section 4.

2.3 Containerisation

Containers are a technology stemming from the concept of operating-system-level virtualization.[6] A container is a lightweight, isolated environment that packages and runs applications with all their dependencies as a self-contained unit in user space, while safely sharing the (Linux) kernel with other containers. This encapsulation facilitates seamless software deployment across diverse computing landscapes. Containers are instantiated from read-only *images* that contain all the necessary components and instructions for creating a container, including application code, runtime platform, libraries, environment variables, and configuration files. An image is defined in a file named `Dockerfile` (or `Containerfile`) using a standard syntax. The task of generating a container image definition for an existing application so it can be run as a container is often referred to as "containerisation".

Containerising an application offers numerous benefits, including scalability, resource efficiency, enhanced security, and improved observability. Since containers share the host operating system's kernel, they incur less overhead compared to traditional virtualisation techniques. This characteristic enables containers to be started and stopped quickly, facilitating rapid scaling of applications to meet fluctuating demands. Containerisation also supports observability akin to bare-metal environments through kernel-level mechanisms such as eBPF[7] and cgroups[8], which enable sophisticated monitoring and resource management. The isolation provided by containers helps prevent conflicts between applications and enhances security by limiting the impact of potential vulnerabilities.

Popular containerisation platforms, e.g., Docker, Podman, LXD, and rkt, have significantly contributed to the widespread adoption of container technology within the modern IT landscape. Notably, Kubernetes (often abbreviated as K8s) has emerged as the de-facto industry standard for container orchestration: automating the deployment, scaling, and management of containerised applications. Kubernetes' YAML-based configuration manifests (JSON-variants are also supported) have become widely adopted as a language for declarative infrastructure-as-code (IaC), enabling developers and operations teams to manage infrastructure through declarative, version-controlled code, rather than through the traditional error-prone mixture of imperative scripts and manual processes. IaC thus facilitates consistent, repeatable, and automated provisioning and deployment of servers, networks, and other infrastructure components.

As an "operating system for the cloud", Kubernetes offers several distributions with varying levels of functionality (and complexity). Nowadays, there exist several lightweight production-ready distributions, e.g., k3s (`k3s.io`), k0s (`k0sproject.io`), and microK8s (`microk8s.io`), that greatly simplify the deployment and management of Kubernetes environments, especially in development, testing, and small-scale production scenarios. These distributions provide an accessible entry point for organizations and individuals looking to adopt Kubernetes without the overhead of its full-scale versions, thus democratizing access to this pivotal technology.

3 Containerising StarExec

StarExec (see Section 2.1) is based around a head node that coordinates activities, in particular the creation of jobs as sets of job pairs, with each pair consisting of an ATP system and a problem file. MariaDB is used to store job information and results, and NFS is used to share disk space between the head node and compute nodes. StarExec currently offers two backends for running

[6]See `en.wikipedia.org/wiki/OS-level_virtualization`

[7]Extended Berkeley Packet Filter, see `en.wikipedia.org/wiki/EBPF`

[8]Linux's control groups, see `en.wikipedia.org/wiki/Cgroups`

job pairs: the local backend that runs pairs on the same computer as the head node, and the Sun Grid Engine (SGE) backend that sends pairs out to compute nodes.

So far, the head node with a local backend has been successfully containerised - see the `starexec-containerised` directory of the GitHub repository. It includes ...

- A `Dockerfile` for building a StarExec image with a local backend.
- A StarExec configuration file (database credentials, special StarExec directory paths, default StarExec users, NFS mount path, etc.).
- Various scripts used in the `Dockerfile` to configure and build StarExec. These scripts are responsible for:
  - Installing and configuring StarExec dependencies including Java, Apache Tomcat, ant, MariaDB, SPSS, and more.
  - Creating new user accounts (at the operating system level) used for running jobs.
  - Changing permissions of certain files and directories that StarExec depends on.
  - Building StarExec using ant, which also initializes the database.
- A `README.md` file explaining how to build and run the image.

The deployment of StarExec Miami was a real challenge, requiring installation and configuration of many pieces of software. The containerisation approach aims to make this process simple and repeatable, eliminating the need to understand the complex environment requirements of StarExec. While the containerisation of StarExec with a local backend is somewhat valuable on its own, it is most importantly a first step towards the deployment of a full StarExec cluster in the cloud. Section 5 explains how this will be done.

4 Containerising ATP Systems

While the grand plan is to deploy ATP systems in a containerised StarExec, and in a Kubernetes hosted version of StarExec, containerising ATP systems is independently useful because it allows ATP systems to be easily deployed in users' applications. It would be great if ATP systems developers become super enthusiastic about containerising their systems after reading this section ☺.

The ATP systems' are containerised in a hierarchy, shown in Figure 2. The underlying operating system is `ubuntu:latest` from `dockerhub` ...

`hub.docker.com/_/ubuntu`

The `ubuntu-arc`[9] container image adds to `ubuntu:latest` using `apt-get` to install common software such as `cmake`, `git`, `tcsh`, `python3`, and `wget`. `ubuntu-arc` also creates an `artifacts` directory where the components required for an ATP system's execution are placed.

The `tptp-world` container image provides utilities from the TPTP World that are used by ATP systems, e.g., `SPCForProblem` detects the Specialist Problem Class (SPC) [42] of a problem that is used by some ATP systems to decide on what search parameters to use. To support these utilities some libraries that are not part of the `ubuntu-arc` have to be added. Additionally, the `runsolver` utility for limiting and reporting the resources used by an ATP system is added. (See Section 4.3 for information about the forthcoming `ResourceLimitRun` utility that will replace `runsolver`.) The details of building the *ATP-system:version* and *ATP-system:version*-`RLR` container images are provided in Section 4.1.

[9]"arc" for "Automated Reasoning Containerisation, or Automated Reasoning in the Cloud".

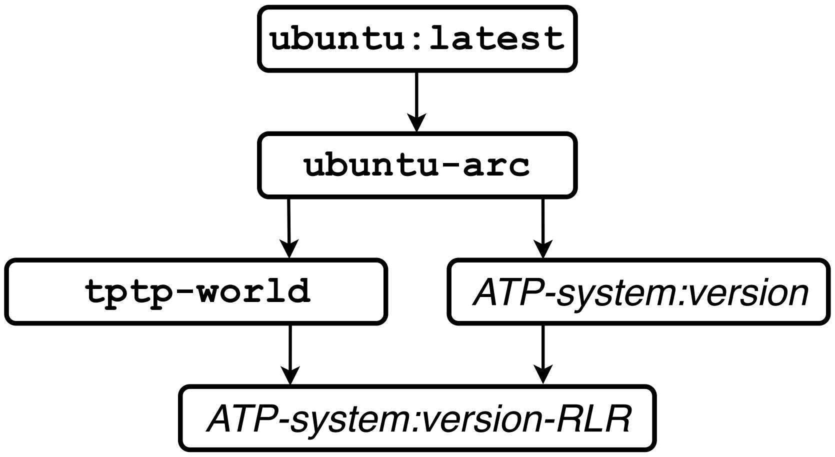

Figure 2: ATP System Container Image Hierarchy

4.1 Building ATP System Containers

Each *ATP-system:version* container image is built on top of the `ubuntu-arc` container image, and with the `tptp-world` container image forms the base for the final *ATP-system:version*-`RLR` container image. The *ATP-system* is the container name, and the *version*/*version*-`RLR` are the container tags. Podman[10] requires the container name to be lowercase, so, e.g., E's container is named `eprover`. The "`RLR`" refers to the "Resource Limited Run" program used to monitor and limit the resources used by the ATP system, either `runsolver` or `ResourceLimitRun`. The files for containerising some ATP systems are in the `provers-containerised` directory of the GitHub repository. A `Makefile` to containerise E, Leo-III, and Vampire is included.

Each *ATP-system:version* container image adds the ATP system's executables to `ubuntu-arc`. The ATP system is retrieved online, e.g., from a GitHub repository, and the necessary commands to build the executables are run. The executables are copied into the `/artifacts` directory. The choice of which version of the ATP system to containerise is made inside the `Dockerfile`. This localization is necessary because the processes for retrieving and building particular ATP system versions vary from system to system and from version to version. An *ATP-system:version* container image must include a `run_system` script to run the ATP system, using whatever incantations are necessary. The parameters for running the ATP system are provided to the `run_system` script in "`RLR`" environment variables (see Section 4.2). Appendix A shows E's `run_system` script. It invokes the `eprover` or `eprover-ho` binary, depending on whether the problem is first-order or higher-order. Depending on the intent, the appropriate command line arguments are given to the selected binary along with the problem file and time limit. Figure 3 shows the `Dockerfile` used to create E's `eprover:3.0.03` container image, using the command "`podman build -t eprover:3.0.03 .`".

Each *ATP-system:version*-`RLR` container image is based on its *ATP-system:version* container image and the `tptp-world` container image. *ATP-system:version*-`RLR` primarily extends `tptp-world`, and copies over only what is necessary from *ATP-system:version*. This simple arrangement allows a generic `Dockerfile` to be used, parameterised by the under-

[10]Our containerisation efforts are carried out using Podman, which is designed to work as a drop-in replacement for Docker (simply aliasing `podman` to `docker` is endorsed in the documentation).

```
#------------------------------------------------------------
FROM ubuntu-arc

# Clones repository
ARG E_VERSION=E-3.0.03
RUN git clone --depth 1 --branch $E_VERSION https://github.com/eprover/eprover.git

# Set working directory to cloned sources directory
WORKDIR /eprover

# Builds first-order executable
RUN ./configure --bindir=/artifacts && \
    make && \
    make install

# Builds higher-order executable
RUN ./configure --enable-ho && \
    make rebuild
RUN cp PROVER/eprover-ho /artifacts/eprover-ho

# run_system script
ADD run_system /artifacts/
#------------------------------------------------------------
```

Figure 3: The `Dockerfile` for E's `-build`

lying *ATP-system:version*. The `ENTRYPOINT` in *ATP-system:version*-`RLR` is the `runsolver` utility from `tptp-world`, which is used to run the ATP system (see Section 4.2). Figure 4 shows the `Dockerfile` to create E's `eprover:3.0.03-RLR` container image, using the command "`podman build -t eprover:3.0.03 RLR --build-arg PROVER_IMAGE=eprover:3.0.03 .`". The *ATP-system:version*-`RLR` container images are pushed to `dockerhub` in ...

`hub.docker.com/repositories/tptpstarexec`

which has a directory for each ATP system. The pushed container images are tagged as *ATP-system-name*`:`*ATP-system-version*-`RLR`-*architecture*, where *architecture* is, e.g., `arm64` or `amd64`.

4.2 Running *ATP-system:version*-RLR Containers

An *ATP-system:version*-`RLR` container image is started using `podman run`. The parameters for running the ATP system are passed into the container in environment variables, using the `-e` option: `RLR_INPUT_FILE` provides the problem file name, `RLR_CPU_LIMIT` provides the CPU time limit in seconds (0 by default, to indicate no limit), `RLR_WC_LIMIT` provides the wall clock time limit in seconds (0 by default, to indicate no limit), `RLR_MEM_LIMIT` provides the memory limit in MiB (0 by default, to indicate no limit), and `RLR_INTENT` indicates the user's intent[11] (`THM` by default). The problem file is passed into the running container using the `-v` option to mount the directory containing the problem file to a directory inside the container,

[11] An *intent* is a tag such as `THM` or `SAT`, indicating that the ATP system should try to prove (or, equivalently for most systems, show that the problem is unsatisfiable) or disprove (or, equivalently for most systems, show that the problem is satisfiable) the conjecture, respectively.

```
#------------------------------------------------------------
ARG PROVER_IMAGE

FROM ${PROVER_IMAGE} AS builder
FROM tptp-world

ENV PATH=".:${PATH}"
WORKDIR /artifacts

# System specific stuff
COPY --from=builder /artifacts/* /artifacts/

ENTRYPOINT ["runsolver"]
#------------------------------------------------------------
```

Figure 4: The generic `Dockerfile` for building `-RLR` container images

and setting the `RLR_INPUT_FILE` environment variable to the name of the problem file in the directory inside the container. The command line parameters for `runsolver` (the `ENTRYPOINT` in the *ATP-system:version*-`RLR` container image) and `run_system` are provided as the remaining parameters to `podman run`. For example, to run the `eprover:3.0.03-RLR` container image on the problem `MGT019+2.p`, the `podman run` could be ...

```
    podman run eprover:3.0.03-RLR -v .:/artifacts/CWD
-e RLR_INPUT_FILE='/artifacts/CWD/MGT019+2.p' -e RLR_CPU_LIMIT='60'
-e RLR_WC_LIMIT='60' -e RLR_MEM_LIMIT='0' -e RLR_INTENT='SAT'
--timestamp -C 60 -W 60 run_system
```

The "`--timestamp -C 60 -W 60`" are command line parameters to `runsolver`.

A Python script `run_image.py` is provided to simplify and standardize running *ATP-system:version*-`RLR` container images. The script is shown in Appendix B. The script must have an *ATP-system:version*-`RLR` container image name as a command line argument. By default `run_image.py` runs the *ATP-system:version*-`RLR` with the problem taken from `stdin`, imposing no CPU, wall clock, or memory limits, with the `THM` intent. All the parameters can be changed with further command line options.

4.3 The `ResourceLimitedRun` Utility

When the TPTP World's SystemOnTPTP service [34] was first made available [35] it used a Perl program called `TreeLimitedRun` to monitor and limit ATP systems' use of CPU time, wall clock time, and memory. As the name suggests, the principle was to monitor the forest of process hierarchies started by an ATP system, understanding that some of the processes might be orphaned and adopted by the `init` process (now `systemd` and others). `TreeLimitedRun` was superseded by `runsolver` [26] that is written in C++, and adopted the same principle for monitoring processes. More recently, `BenchExec` [3], which is used in StarExec Iowa, has taken advantage of Linux's cgroup v2 subsystem, which provides operating system level support for monitoring processes. `BenchExec` is written in Python, with rather heavy installation requirements. The new `ResourceLimitedRun` utility is written in C, and also uses Linux's cgroup v2 subsystem. `ResourceLimitedRun` has the same command line parameters as `runsolver`, and thus can be substituted for `runsolver` (and `BenchExec`). `ResourceLimitedRun` is being tested at the time of writing, and hopefully will be deployed at the time of presentation!

5 Towards a Cloud-native StarExec

The term "cloud-native" has increasingly become synonymous with an approach to designing and operating applications that fully leverage the benefits of the cloud computing model.[12] Cloud-native applications are distinguished by their ability to scale effectively, utilising the cloud's capability to dynamically allocate resources. This development paradigm is closely aligned with DevOps practices that emphasize collaboration between development and operations teams to automate the process of software delivery and infrastructure changes. It inherently supports infrastructure-as-code (IaC), a key DevOps practice, enabling the management and provisioning of infrastructure through declarative, version-controlled definition files that are both human- and machine-readable.

Containers (see Section 2.3) play a pivotal role in cloud-native development. `Dockerfile`s are used to specify the steps to create a container image, embodying the IaC philosophy by detailing the desired (partial) state of a containerised application. Similarly, Kubernetes YAML manifests define, in a declarative fashion, how application components are deployed and run on Kubernetes clusters, aligned with the IaC paradigm.

The synthesis of these practices allows for the entire stack, from infrastructure to application, to be declaratively specified, versioned, and automatically deployed as required. The reliance on mainstream open source technologies such as the CNCF Kubernetes and Podman projects (`www.cncf.io/projects`) offers unparalleled flexibility, scalability, and portability, free from the constraints of single vendors or platforms. An open distribution model ensures that StarExec's infrastructure "as code" is readily accessible for modification and distribution, e.g., by cloning or forking from our GitHub repository. Adopting these technologies will allow ATP systems and StarExec, including the requisite infrastructure, to be deployed by ATP system developers and users in their preferred cloud environment or even in on-premises servers. This approach significantly simplifies the process of utilizing state-of-the-art ATP technology, making it much more easily usable by anyone, anywhere.

5.1 Re-engineering StarExec for the Cloud

Recalling the current architecture of StarExec described in Section 3, several areas for improvement have been identified to better serve the needs for re-engineering. Our ongoing efforts include:

1. **Utilization of containerised ATP systems** (see Section 4), which will be hosted in a publicly accessible container image registry, instead of the current approach of requiring StarExec users to build and upload a StarExec `.tgz` package according to StarExec specifications.

2. **Adding an abstraction layer for database communication** with the relational database (currently MariaDB) used to persist job information. This layer will allow the database component to operate in its own container, significantly reducing coupling. Furthermore, by eliminating MariaDB-specific bindings, compatibility with other database systems will be enabled. This flexibility allows for seamless integration with existing SQL databases within the user's infrastructure, enhancing portability and adaptability.

[12]For more information refer to the initiatives led by the Cloud Native Computing Foundation (CNCF) at `www.cncf.io`, which advocates for the adoption of this paradigm by fostering and sustaining an ecosystem of open source, vendor-neutral projects.

3. **Using Kubernetes job scheduling facilities**, thereby completely replacing the current SGE cluster management. This change offers numerous benefits:
   - **Scalability:** Kubernetes excels at managing and scaling containerised applications, adapting to fluctuating workloads with ease. It also seamlessly integrates with most infrastructure provisioning tools, supporting both cloud and on-premise platforms.
   - **Monitoring:** A vast array of observability tools (encompassing logging, metrics, tracing, etc.) support seamless integration with Kubernetes. Additionally, with its self-healing features, Kubernetes can automatically restart failed containers, replace and reschedule containers when nodes die, and kill non-responsive containers.
   - **Efficiency:** Similar to SGE and other cluster management software such as Slurm and Torque, Kubernetes optimizes the use of underlying hardware by efficiently scheduling jobs and managing resources.[13]
   - **Flexibility:** Kubernetes' extensible architecture allows for custom schedulers and automated scaling decisions, enabling it to support a wide range of workloads, including stateless, stateful, and batch processing.

Figure 5 shows a generic architecture of the future re-engineered StarExec using Kubernetes.

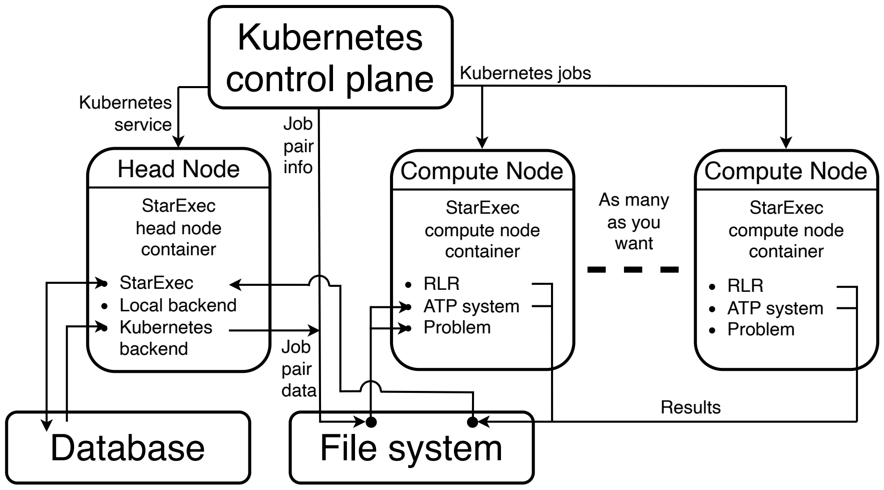

Figure 5: Projected StarExec generic architecture

5.2 StarExec in AWS

An Amazon Research Award[14] has been granted to deploy StarExec in AWS. This will not only help fund the development efforts discussed in Section 5.1, but will also fund a first fully-

[13]Certainly, Kubernetes does not outperform traditional High-Performance Computing (HPC) software within their specialized application domains. Kubernetes' extensible architecture facilitates interfacing with HPC systems through custom schedulers if the need arises (see `kubernetes.io/docs/concepts/extend-kubernetes`). In this setup, Kubernetes oversees container orchestration, while delegating the scheduling of intensive computing tasks to specialized HPC software.

[14]Amazon Research Award, Fall 2023. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors, and do not reflect the views of Amazon.

functional reference deployment of StarExec in the AWS cloud. The generic architecture in Figure 5 will be instantiated using concrete AWS-managed services, as shown in Figure 6.

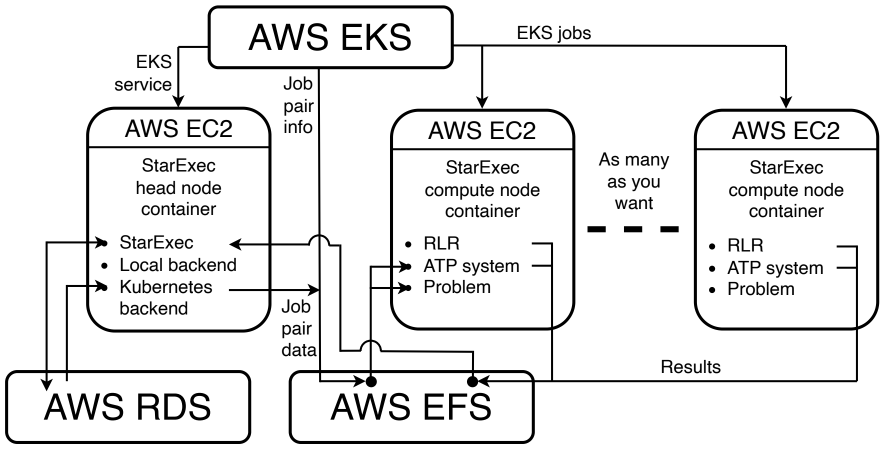

Figure 6: Architecture in AWS

- The Kubernetes control plane will be managed by AWS Elastic Kubernetes Service (EKS).
- The StarExec head and compute nodes will run on suitable Amazon EC2 instances, currently planned to be `x2iedn.xlarge` instances that have four Intel Xeon Scalable vCPUs running up to 3.5GHz, and 128 GiB memory.
- The database will be Amazon Relational Database (RDS).
- The file system will be Amazon Elastic File System (EFS).
- The ATP systems' containerisation can be made compatible with (possibly be exactly) the Amazon Trusted Solver format, as was recently used in the SMT and SAT competitions[15].

Leveraging AWS-managed services will expedite the delivery of StarExec's initial cloud-native version to the community. This approach will particularly benefit teams planning to deploy StarExec on their own AWS accounts or through AWS grants. The initial release will be rigorously tested through the migration of the TPTP community from StarExec Miami to the new StarExec AWS platform. We are particularly enthusiastic about collaborating with teams interested in deploying StarExec on their on-premise infrastructure or within university HPC clusters.

6 Conclusion

This paper has described work being done to containerise StarExec and ATP systems so that they can be run on a broad range of computer platforms. Additionally, this work explains plans to build backend in StarExec so that Kubernetes can be used to orchestration distribute of StarExec job pairs over whatever compute nodes are available.

This is ongoing work – some of the work is still in progress, particularly embedding StarExec in Kubernetes on AWS. Hopefully the future will include StarExec being flexibly available in online compute clusters.

[15] `github.com/aws-samples/aws-batch-comp-infrastructure-sample`

References

[1] E. Bartocci, D. Beyer, P.E. Black, G. Fedyukovich, H. Garavel, A. Hartmanns, M. Huisman, F. Kordon, J. Nagele, M. Sighireanu, B. Steffen, M. Suda, G. Sutcliffe, T. Weber, and A. Tamada. TOOLympics 2019: An Overview of Competitions in Formal Methods. In T. Vojnar and L. Zhang, editors, *Proceedings of the 2019 International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, number 11429 in Lecture Notes in Computer Science, pages 3–24. Springer-Verlag, 2019.

[2] C. Benzmüller and B. Woltzenlogel Paleo. Automating Gödel's Ontological Proof of God's Existence with Higher-order Automated Theorem Provers. In T. Schaub, editor, *Proceedings of the 21st European Conference on Artificial Intelligence*, pages 93–98, 2014.

[3] D. Beyer, S. Löwe, and P. Wendler. Reliable Benchmarking: Requirements and Solutions. *International Journal on Software Tools for Technology Transfer*, 21:1–29, 2019.

[4] A. Bruni, E. Drewsen, and C. Schürmann. Towards a Mechanized Proof of Selene Receipt-Freeness and Vote-Privacy. In R. Krimmer, M. Volkamer, N. Braun Binder, N. Kersting, O. Pereira, and C. Schürmann, editors, *Proceedings of the International Joint Conference on Electronic Voting, E-Vote-ID 2017*, number 10615 in Lecture Notes in Computer Science, pages 110–126. Springer-Verlag, 2017.

[5] M. Caminati, M. Kerber, C. Lange, and C. Rowat. Sound Auction Specification and Implementation. In M. Feldman, M. Schwarz, and T. Roughgarden, editors, *Proceedings of the 16th ACM Conference on Economics and Computation*, pages 547–564. ACM Press, 2015.

[6] V. Chaudri, B. Cheng, A. Overholtzer, J. Roschelle, A. Spaulding, P. Clark, M. Greaves, and D. Gunning. Inquire Biology: A Textbook that Answers Questions. *AI Magazine*, 34(3), 2013.

[7] D. Cok, A. Stump, and T. Weber. The 2013 Evaluation of SMT-COMP and SMT-LIB. *Journal of Automated Reasoning*, 55(1):61–90, 2015.

[8] B. Cook. Formal Reasoning About the Security of Amazon Web Services. In H. Chockler and G. Weissenbacher, editors, *Proceedings of the 30th International Conference on Computer Aided Verification*, number 10981 in Lecture Notes in Computer Science, pages 38–47. Springer-Verlag, 2018.

[9] L. Dennis, M. Fisher, M. Slavkovik, and M. Webster. Formal Verification of Ethical Choices in Autonomous Systems. *Robotics and Autonomous Systems*, 77:1–14, 2016.

[10] R. Hähnle and M. Huisman. Deductive Software Verification: From Pen-and-Paper Proofs to Industrial Tools. In B. Steffen and G. Woeginger, editors, *Computing and Software Science: State of the Art and Perspectives*, number 10000 in Lecture Notes in Computer Science, pages 345–373. Springer-Verlag, 2019.

[11] M.T. Hannan. Rethinking Age Dependence in Organizational Mortality: Logical Formalizations. *American Journal of Sociology*, 104:126–164, 1998.

[12] J. Harrison. Floating-Point Verification using Theorem Proving. In M. Bernardo and A. Cimatti, editors, *Proceedings of the 6th International School on Formal Methods for the Design of Computer, Communication, and Software Systems*, number 3965 in Lecture Notes in Computer Science, pages 211–242. Springer-Verlag, 2006.

[13] S. Holden. Connect++: A New Automated Theorem Prover Based on the Connection Calculus. In J. Otten and W. Bibel, editors, *Proceedings of the 1st International Workshop on Automated Reasoning with Connection Calculi*, number 3613 in CEUR Workshop Proceedings, pages 95–106, 2023.

[14] A. Hommersom, P. Lucas, and P. van Bommel. Automated Theorem Proving for Quality-checking Medical Guidelines. In G. Sutcliffe, B. Fischer, and S. Schulz, editors, *Proceedings of the Workshop on Empirically Successful Classical Automated Reasoning*, 2005.

[15] H. Hoos and T. Stützle. SATLIB: An Online Resource for Research on SAT. In I. Gent, H. van Maaren, and T. Walsh, editors, *Proceedings of the 3rd Workshop on the Satisfiability Problem*,

pages 283–292. IOS Press, 2000.

[16] J. Horner. A Computationally Assisted Reconstruction of an Ontological Argument in Spinoza's The Ethics. *Open Philosophy*, 2:219–229, 2019.

[17] K. Korovin. Implementing an Instantiation-based Theorem Prover for First-order Logic. In C. Benzmüller, B. Fischer, and G. Sutcliffe, editors, *Proceedings of the 6th International Workshop on the Implementation of Logics*, number 212 in CEUR Workshop Proceedings, pages 63–63, 2006.

[18] T. Libal. Towards Automated GDPR Compliance Checking. In F. Heintz, M. Milano, and B. O'Sullivan, editors, *Proceedings of the International Workshop on the Foundations of Trustworthy AI Integrating Learning, Optimization and Reasoning*, number 12641 in Lecture Notes in Computer Science, pages 3–19, 2020.

[19] C. Marché and H. Zantema. The Termination Competition. In F. Baader, editor, *Proceedings of the 18th International Conference on Term Rewriting and Applications*, number 4533 in Lecture Notes in Computer Science, pages 303–313, 2007.

[20] W.W. McCune. Otter 3.3 Reference Manual. Technical Report ANL/MSC-TM-263, Argonne National Laboratory, Argonne, USA, 2003.

[21] T. Nipkow. Social Choice Theory in HOL: Arrow and Gibbard-Satterthwaite. *Journal of Automated Reasoning*, 43(3):289–304, 2009.

[22] P. Oppenheimer and E. Zalta. A Computationally-Discovered Simplification of the Ontological Argument. *Australasian Journal of Philosophy*, 89(2):333–349, 2011.

[23] J. Otten. 20 Years of leanCoP - An Overview of the Provers. In J. Otten and W. Bibel, editors, *Proceedings of the 1st International Workshop on Automated Reasoning with Connection Calculi*, number 3613 in CEUR Workshop Proceedings, pages 4–22, 2023.

[24] H. Prakken and G. Sartor. Law and Logic: A Review from an Argumentation Perspective. *Artificial Intelligence*, 227:214–245, 2015.

[25] A. Riazanov and A. Voronkov. The Design and Implementation of Vampire. *AI Communications*, 15(2-3):91–110, 2002.

[26] O. Roussel. Controlling a Solver Execution with the `runsolver` Tool. *Journal of Satisfiability, Boolean Modeling and Computation*, 7(4):139–144, 2011.

[27] S. Schulz. Algorithms and Data Structures for First-Order Equational Deduction. In C. Benzmüller, B. Fischer, and G. Sutcliffe, editors, *Proceedings of the 6th International Workshop on the Implementation of Logics*, number 212 in CEUR Workshop Proceedings, pages 1–6, 2006.

[28] S. Schulz. Simple and Efficient Clause Subsumption with Feature Vector Indexing. In M.P. Bonacina and M. Stickel, editors, *Automated Reasoning and Mathematics: Essays in Memory of William W. McCune*, number 7788 in Lecture Notes in Artificial Intelligence, pages 45–67. Springer-Verlag, 2013.

[29] S. Schulz, S. Cruanes, and P. Vukmirović. Faster, Higher, Stronger: E 2.3. In P. Fontaine, editor, *Proceedings of the 27th International Conference on Automated Deduction*, number 11716 in Lecture Notes in Computer Science, pages 495–507. Springer-Verlag, 2019.

[30] S. Schulz and A. Pease. Teaching Automated Theorem Proving by Example: PyRes 1.2 (system description). In N. Peltier and V. Sofronie-Stokkermans, editors, *Proceedings of the 10th International Joint Conference on Automated Reasoning*, number 12167 in Lecture Notes in Computer Science, pages 158–166, 2020.

[31] A. Steen. Scala TPTP Parser v1.5, 2021. DOI: 10.5281/zenodo.5578872.

[32] A. Steen and C. Benzmüller. The Higher-Order Prover Leo-III. In D. Galmiche, S. Schulz, and R. Sebastiani, editors, *Proceedings of the 8th International Joint Conference on Automated Reasoning*, number 10900 in Lecture Notes in Artificial Intelligence, pages 108–116, 2018.

[33] A. Stump, G. Sutcliffe, and C. Tinelli. StarExec: a Cross-Community Infrastructure for Logic Solving. In S. Demri, D. Kapur, and C. Weidenbach, editors, *Proceedings of the 7th International Joint Conference on Automated Reasoning*, number 8562 in Lecture Notes in Artificial Intelligence,

pages 367–373, 2014.

[34] G. Sutcliffe. SystemOnTPTP. In D. McAllester, editor, *Proceedings of the 17th International Conference on Automated Deduction*, number 1831 in Lecture Notes in Artificial Intelligence, pages 406–410. Springer-Verlag, 2000.

[35] G. Sutcliffe. TPTP, TSTP, CASC, etc. In V. Diekert, M. Volkov, and A. Voronkov, editors, *Proceedings of the 2nd International Symposium on Computer Science in Russia*, number 4649 in Lecture Notes in Computer Science, pages 6–22. Springer-Verlag, 2007.

[36] G. Sutcliffe. The SZS Ontologies for Automated Reasoning Software. In G. Sutcliffe, P. Rudnicki, R. Schmidt, B. Konev, and S. Schulz, editors, *Proceedings of the LPAR Workshops: Knowledge Exchange: Automated Provers and Proof Assistants, and the 7th International Workshop on the Implementation of Logics*, number 418 in CEUR Workshop Proceedings, pages 38–49, 2008.

[37] G. Sutcliffe. The TPTP World - Infrastructure for Automated Reasoning. In E. Clarke and A. Voronkov, editors, *Proceedings of the 16th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning*, number 6355 in Lecture Notes in Artificial Intelligence, pages 1–12. Springer-Verlag, 2010.

[38] G. Sutcliffe. The CADE ATP System Competition - CASC. *AI Magazine*, 37(2):99–101, 2016.

[39] G. Sutcliffe. The TPTP Problem Library and Associated Infrastructure. From CNF to TH0, TPTP v6.4.0. *Journal of Automated Reasoning*, 59(4):483–502, 2017.

[40] G. Sutcliffe. Stepping Stones in the TPTP World. In C. Benzmüller, M. Heule, and R. Schmidt, editors, *Proceedings of the 12th International Joint Conference on Automated Reasoning*, number 14739 in Lecture Notes in Artificial Intelligence, pages 30–50, 2024.

[41] G. Sutcliffe, S. Schulz, K. Claessen, and A. Van Gelder. Using the TPTP Language for Writing Derivations and Finite Interpretations. In U. Furbach and N. Shankar, editors, *Proceedings of the 3rd International Joint Conference on Automated Reasoning*, number 4130 in Lecture Notes in Artificial Intelligence, pages 67–81. Springer, 2006.

[42] G. Sutcliffe and C.B. Suttner. Evaluating General Purpose Automated Theorem Proving Systems. *Artificial Intelligence*, 131(1-2):39–54, 2001.

[43] A' Voronkov. Algorithms, Datastructures, and Other Issues in Efficient Automated Deduction. In R. Gore, A. Leitsch, and T. Nipkow, editors, *Proceedings of the International Joint Conference on Automated Reasoning*, number 2083 in Lecture Notes in Artificial Intelligence, pages 13–28. Springer-Verlag, 2001.

[44] M. Yadav. On the Synthesis of Machine Learning and Automated Reasoning for an Artificial Synthetic Organic Chemist. *New Journal of Chemistry*, 41(4):1411–1416, 2017.

A E's run_system script

```
#------------------------------------------------------------------------------
#!/bin/tcsh

setenv HERE `dirname $0`
setenv TEMPDIR `mktemp -d`
setenv PROBLEMFILE $TEMPDIR/E---3.1_$$.p
onintr cleanup

#----Add extra ()s for THF and TXF
$HERE/tptp4X -t uniquenames4 -x $RLR_INPUT_FILE > $PROBLEMFILE

set SPCLine=`grep -E "^% SPC " $PROBLEMFILE`
if ("$SPCLine" != "") then
    set ProblemSPC = `expr "$SPCLine" : "^% SPC  *: *\([^ ]*\)"`
else
    set ProblemSPC = `$HERE/SPCForProblem $RLR_INPUT_FILE`
endif
set Mode = $RLR_INTENT

set CommonParameters="--delete-bad-limit=2000000000 --definitional-cnf=24 \
-s --print-statistics -R --print-version --proof-object --cpu-limit=$RLR_WC_LIMIT"
if ("$Mode" == "THM") then
    if (`expr "$ProblemSPC" : "TH0_.*"`) then
        echo "Running higher-order theorem proving"
        $HERE/eprover-ho $CommonParameters --auto-schedule=8 $PROBLEMFILE
    else
        echo "Running first-order theorem proving"
        $HERE/eprover $CommonParameters --auto-schedule=8 $PROBLEMFILE
    endif
else
    echo "Running first-order model finding"
    $HERE/eprover $CommonParameters --satauto-schedule=8 $PROBLEMFILE
endif

cleanup:
    echo "% E exiting"
    rm -rf $TEMPDIR
#------------------------------------------------------------------------------
```

B run_image.py

```
#-------------------------------------------------------------------------------
#!/usr/bin/env python3

import argparse
import subprocess
import os, sys
import shutil

def getRLRArgs(args):
    mem_part = f" -M {args.memory_limit}" if args.memory_limit > 0 else ""
    return "--timestamp --watcher-data /dev/null -C " + \
f"{args.cpu_limit} -W {args.wall_clock_limit}{mem_part}"

def getEnvVars(args):
    return " ".join([f"-e {k}='{v}'" for k, v in [
        ("RLR_INPUT_FILE", "/artifacts/CWD/problemfile"),
        ("RLR_CPU_LIMIT", args.cpu_limit), ("RLR_WC_LIMIT", args.wall_clock_limit),
        ("RLR_MEM_LIMIT", args.memory_limit), ("RLR_INTENT", args.intent),
    ]])

def makeBenchmark(problem):
    if problem:
        shutil.copy(problem, "./problemfile")
    else:
        with open('./problemfile', 'w') as problemfile:
            problemfile.write(sys.stdin.read())

if __name__ == "__main__":
    parser = argparse.ArgumentParser("Wrapper for a podman call to a prover image")
    parser.add_argument("image_name",
help="Image name, e.g., eprover:3.0.03-RLR-arm64")
    parser.add_argument("-P", "--problem",help="Problem file if not stdin")
    parser.add_argument("-C", "--cpu-limit", default=0, type=int,
help="CPU time limit in seconds, default=none")
    parser.add_argument("-W", "--wall-clock-limit", default=0, type=int,
help="Wall clock time limit in seconds, default=none")
    parser.add_argument("-M", "--memory-limit", default=0, type=int,
help="Memory limit in MiB, default=none")
    parser.add_argument("-I", "--intent", default="THM", choices=["THM", "SAT"],
help="Intention (THM, SAT, etc), default=THM")
    args = parser.parse_args()
    if args.wall_clock_limit == 0 and args.cpu_limit != 0:
        args.wall_clock_limit = args.cpu_limit

    command = f"podman run {getEnvVars(args)} -v .:/artifacts/CWD -t " + \
f"{args.image_name} {getRLRArgs(args)} run_system"
    makeBenchmark(args.problem)
    subprocess.run(command, shell=True)
    os.remove("./problemfile")
#-------------------------------------------------------------------------------
```