Running Grid Jobs in Containers
A new approach to job execution for the next generation Grid framework

Maxim.Storetvedt@hvl.no  | CHEP 2019 - Track 7: Facilities, Clouds and Containers  | 05/11/2019
The ALICE collaboration is moving towards a new Grid middleware framework: **jAliEn** (The Java ALICE Environment)

- Successor to the ALICE Environment (AliEn), which is currently used in production

Several major changes from AliEn

- Written entirely in **Java**
- More efficient DB backends
- Load balancing
- **New authentication scheme**
  - Based around X.509 token certificates
jAliEn Token Certificates

• Time limited X.509 certificates representing a role
  • Full X.509, signed by internal CA
  • Identities divided into groups, each representing a Grid role
  • Predefined set of privileges and limitations defined for each group
• Allows for assigning **fine grained permissions to each role**

Token roles and permissions:

**Users**: Same permissions as the identity certificate of the user

**Jobs**: Permissions to run job payload, mainly read-only file operations

**JobAgents**: Permissions to do job matching, update job status and traces
The jAliEn Grid Job Pilot

- Initial jAliEn job-pilot modeled after the "original" AliEn
  - Does both job matching, and job execution
  - Consequently, runs with both a Job and a JobAgent token
The jAliEn Grid Job Pilot

- Initial jAliEn job-pilot modeled after the "original" AliEn
  - Does both job matching, and job execution
  - Consequently, runs with both a Job and a JobAgent token
- Arguably defeats the purpose of having fine-grained token certificates
A new Grid job pilot for jAliEn

- **Goal**: Fully harness the presence of fine-grained token certificates to introduce more isolation within Grid job execution
• **Goal:** Fully harness the presence of fine-grained token certificates to introduce more isolation within Grid job execution

• **Idea:** Split the job pilot into two "layers"
  - Let the process matching Grid jobs run with its own token (the Job Agent)
  - Let the Grid job run in a wrapper with its own token (the Job Wrapper)
A new Grid job pilot for jAliEn

- The two separate layers act as two completely independent processes
  - Each running on its own JVM
A new Grid job pilot for jAliEn

• The two separate layers act as **two completely independent processes**
  • Each running on its own JVM
• With the "old" AliEn, Virtual Machines (VMs) may be used to
  • Separate the payload/JobAgent from the host
  • Simplify configuration
A new Grid job pilot for jAliEn

• The two separate layers act as **two completely independent processes**
  • Each running on its own JVM
• With the "old" AliEn, Virtual Machines (VMs) may be used to
  • Separate the payload/JobAgent from the host
  • Simplify configuration
• With the wrapper being its own process, we could put that in its **own container**
  • Avoids any virtualisation overhead, while still
    • Increasing isolation
    • Providing a cohesive execution environment
Using the new job pilot
Using the new job pilot

Batch Queue

Startup script with embedded agent token

C=ch/O=AliEn/CN=JobAgent
Using the new job pilot

Batch Queue

Startup script with embedded agent token

C=ch/O=AliEn/CN=JobAgent

Starts

Job-Agent

Agent token

*JobAgent may also be containerised (by site)
Using the new job pilot

Startup script with embedded agent token

Receive job ID, JDL and job token

getMatchJob()

Container*

Agent token

*JobAgent may also be containerised (by site)
Using the new job pilot

Batch Queue

Startup script with embedded agent token

C=ch/O=AliEn/CN=JobAgent

getMatchJob()

Receive job ID, JDL and job token

C=ch/O=AliEn/CN=Jobs/CN=<owner>/OU=<role>/OU=<jobId>

*JobAgent may also be containerised (by site)
Using the new job pilot

Batch Queue

Startup script with embedded agent token

C=ch/O=AliEn/CN=JobAgent

getMatchJob()

Receive job ID, JDL and job token

C=ch/O=AliEn/CN=Jobs/CN=<owner>/OU=<role>/OU=<jobId>

*JobAgent may also be containerised (by site)
The problem with layer communications

• The agent and wrapper processes must communicate somehow
  • Separate JVMs
  • May or may not be in a container
  • And what about security?
The problem with layer communications

• The agent and wrapper processes must communicate somehow
  • Separate JVMs
  • May or may not be in a container
  • And what about security?
• Using files and shared directories is not an option
  • Not secure
  • FS may be networked → slow / unreliable
The problem with layer communications

- The agent and wrapper processes must communicate somehow
  - Separate JVMs
  - May or may not be in a container
  - And what about security?
- Using files and shared directories is not an option
  - Not secure
  - FS may be networked → slow / unreliable
- This includes using Unix sockets
  - Will also depend on FS
Chosen solution

• **Idea**: A parent process has access to the **STDIN/STDOUT** of its child
  • The agent process is a parent of the wrapper, and can access its file descriptors (/proc)
Chosen solution

• **Idea:** A parent process has access to the **STDIN/STDOUT** of its child
  • The agent process is a parent of the wrapper, and can access its file descriptors (/proc)
  • Use these to create a data "pipe":
    • Serialise and send to **STDIN**
    • Read from **STDOUT**
Chosen solution

• **Idea:** A parent process has access to the **STDIN/STDOUT** of its child
  • The agent process is a parent of the wrapper, and can access its file descriptors (/proc)
  • Use these to create a data "pipe":
    • Serialise and send to **STDIN**
    • Read from **STDOUT**
• But what if the wrapper is in a container?
  • File descriptors are handled by the **kernel**
    • **Shared** by the container
Vendor lock-in concerns

• Containers remain a relatively new concept
  • Frequent changes within container platforms
  • Frequent changes to software requirements
  • New solutions and features continuously being introduced
• But we are "forced" to target a specific release in production
  • How to remain open for changes?
  • How to avoid being locked to a certain vendor?
Towards "vendor freedom"

• Keeping platform specifics to a minimum is key!
Towards "vendor freedom"

- Keeping platform specifics to a minimum is key!
  - CVMFS can be of great help here
- Single generic call to container service\(^1\) once, thereafter:
  - Container environment init from CVMFS
  - Generic symlink to images in CVMFS
  - Files and packages from CVMFS
    - Even the container service can run directly from CVMFS
  - Job specific data piped from the JobAgent

\(^1\)Singularity, Docker, Podman, etc
Towards "vendor freedom"

• Keeping platform specifics to a minimum is key!
  • CVMFS can be of great help here
• Single generic call to container service¹ once, thereafter:
  • Container environment init from CVMFS
  • Generic symlink to images in CVMFS
  • Files and packages from CVMFS
    • Even the container service can run directly from CVMFS
  • Job specific data piped from the JobAgent
• Everything managed from the wrapper process running inside the container

¹Singularity, Docker, Podman, etc
Our current approach

• Full init flow for our current setup (*Singularity*):
  • JobAgent call to Singularity w/ image in CVMFS
  • Container launches, initialises environment from CVMFS
  • Starts the JobWrapper
    • Both Java runtime and JAR in CVMFS
  • JobAgent "talks" directly to wrapper using STDIN/STDOUT
    • Sends credentials, JDL and data
  • The wrapper will thereafter
    • Download necessary files
    • Run the payload
Summary

- **jAliEn**: A new Grid middleware is being introduced within the ALICE collaboration
- Using the **new features** of jAliEn, a new **two-layer job pilot** is being created
  - Fine-grained control of permissions
  - Executes Grid jobs in **containers**
- One layer for job matching, and second inside the container as a job payload wrapper
  - Communication between layers using an **STDIN/STDOUT pipe**
  - **Avoids** reliance on file system for security/compatibility
- Can **support any container platform**
  - Once a container is launched, the **wrapper process** will do the rest
  - Relies heavily on **CVMFS**
- Currently testing in preparation for Run3
  - [https://gitlab.cern.ch/jalien/jalien](https://gitlab.cern.ch/jalien/jalien)
Background: Why jAliEn?

- Rising concerns the software stack used for the past 10+ years will not be able to scale to the needs of the collaboration
  - An almost 10x increase from 2011 to 2019
  - ALICE detector upgrade for Run3 will greatly increase the amount of collected data
- Maintaining the current Grid middleware (AliEn) is already proving to be a challenge
  - Patchwork of modules and features
Background: Why jAliEn?

- Rising concerns the software stack used for the past 10+ years will not be able to scale to the needs of the collaboration
  - An almost 10x increase from 2011 to 2019
  - ALICE detector upgrade for Run3 will greatly increase the amount of collected data
- Maintaining the current Grid middleware (AliEn) is already proving to be a challenge
  - Patchwork of modules and features

A complete, up-to-date, rewrite can solve these problems
  - No risk of inheriting old "hacks" and "quick fixes"
  - Allows for greater architectural changes and streamlining
    - Straightforward to add new features
Compatibility

• Tested and working with most popular platforms
  • Singularity ✓
  • Docker ✓
  • Rkt ✓
  • Podman ✓
  • Kata ✗

• The exception being Kata Containers
  • (In reality, tiny virtual machines with their own kernel)