ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-WEDPL02

AliECS: A NEW EXPERIMENT CONTROL SYSTEM FOR THE ALICE EXPERIMENT

T. Mrnjavac*, K. Alexopoulos†, V. Chibante Barroso‡, G. Raduta§, CERN, Geneva, Switzerland

Abstract

The ALICE Experiment at CERN LHC (Large Hadron Collider) is undertaking during Long Shutdown 2 in 2019-2020 a major upgrade, which includes a new computing system called O² (Online-Offline). To ensure the efficient operation of the upgraded experiment along with its newly designed computing system, a reliable, high performance and automated experiment control system is being developed with the goal of managing all O² synchronous processing software, and of handling the data taking activity by interacting with the detectors, the trigger system and the LHC. The ALICE Experiment Control System (AliECS) is a distributed system based on state of the art cluster management and microservices which have recently emerged in the distributed computing ecosystem. Such technologies will allow the ALICE Collaboration to benefit from a vibrant and innovating open source community. This communication illustrates the AliECS architecture. It provides an in-depth overview of the system's components, features and design elements, as well as its performance. It also reports on the experience with AliECS as part of ALICE Run 3 detector commissioning setups.

INTRODUCTION

The O^2 Computing System

The ALICE experiment [1] is undergoing a major upgrade [2] which is being deployed during LHC's Long Shutdown 2 (2019-2020) in preparation for LHC Run 3. The new and upgraded detectors result in a significantly increased data rate, and in order for the data processing to keep up, a new computing system called O^2 [3] is being designed, developed and deployed.

In its production stage, the O^2 computing system will consist of 100,000s of processes, deployed over roughly 1000 heterogeneous nodes, fulfilling roles including data readout, processing, storage and auxiliary services. The system will read out 27 Tb/s of raw data and record 800 Gb/s of reconstructed data.

The data driven components of the O^2 computing system will run on two main typologies of computing nodes: FLPs (First Level Processors) and EPNs (Event Processing Nodes). Each FLP is fitted with CRU (Common Readout Unit) [4] or C-RORC (Common Readout Receiver Card) [5] hardware, depending on the detector. These PCI-Express cards are capable of two way communication with detector front end electronics.

* e-mail: teo.m@cern.ch
† e-mail: kostas.alexopoulos@cern.ch
‡ e-mail: vmcb@cern.ch
§ e-mail: george.raduta@cern.ch

The O^2 computing system will be capable of two kinds of data-driven workflows: synchronous operation, intended to be *synchronous* with detector readout, and asynchronous operation, which will take place at any time regardless of detector or beam conditions. Most nodes are expected to run dozens of processes of different kinds, including long running services, WLCG-like (Worldwide LHC Computing Grid) environments for asynchronous processing, and data-driven process workflows. Synchronous workflows operate on data coming from detector data links, thus they must run in the O^2 facility at LHC Point 2. Asynchronous workflows do not have this constraint, they can therefore run at any time on WLCG nodes, or on O^2 facility resources when they are not needed for synchronous operation.

The O^2/FLP Computing Cluster

O^2 is being developed as a complete solution for the data processing needs of the ALICE experiment during Run 3, but the O^2 compute system is split up in two separate computing clusters due to significant differences in requirements between FLPs and EPNs. This division yields the O^2/FLP computing cluster and the O^2/EPN computing cluster, both deployed at LHC Point 2.

The fundamental difference between these two main kinds of nodes stems from the fact that FLPs have direct fiber links to detector front end electronics, making them permanently bound to a specific detector or detector component. Different FLPs may also have a variable number of CRU or C-RORC cards, and different system specifications. FLPs are not interchangeable, thus the O^2/FLP cluster is inevitably a heterogeneous environment. On the other hand, EPNs do not have direct links to detector front end electronics, and they are largely interchangeable, with the purpose of hosting scalable processing workflows which can be replicated on as many EPNs as needed, depending on the required workload.

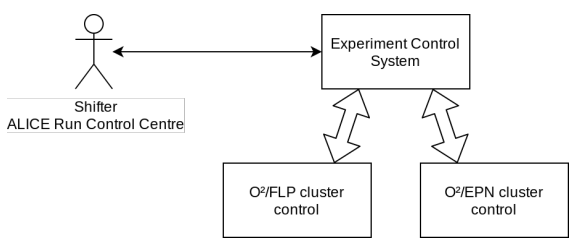

Figure 1: O^2/FLP and O^2/EPN cluster control with respect to the ALICE Run Control Centre.

Each of the two computing clusters has its own specialized control system. Ultimately, the O^2 system as a whole will be controlled via a single user interface, an ECS (Experiment Control System) solution in the ALICE Run Control Centre (see Fig. 1).

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-WEDPL02

Target Operational Improvements in an Experiment Control System for ALICE Run 3

The goals and requirements of the ALICE experiment control system (AliECS) are derived from experience in running the previous computing system (LHC Run 1 and 2) [6], and are motivated by a desire for greater reliability, performance, maintainability, and operational flexibility.

Target operational improvements include

1. no workflow redeployment when including or excluding a detector from data taking,
2. recovery from process and server crashes,
3. process reconfiguration without mandatory restart,
4. and EPN scaling during data taking (e.g. as luminosity decreases towards the end of a LHC fill).

The O^2 project includes a redesign of user interfaces, in favor of next-generation web-based GUIs with SSO (single sign-on) and a revamped design. AliECS comes with a number of command line and graphical user interfaces, including shifter oriented GUIs which supersede those of the previous generation ECS.

Finally, the O^2 project is an opportunity to take advantage of modern developments in computing, thus AliECS is built with the best practices of a microservices distributed application paradigm, and harnessing the features of modern cluster resource management systems.

REQUIREMENTS OF AN ECS SOLUTION FOR ALICE RUN 3

AliECS is a new control system, currently under development with the goal of managing the O^2 facility, and of handling the various phases of the data taking activity by interfacing with the detectors, the trigger system and the LHC. AliECS includes common large high-energy physics experiment control functionalities, such as configuration and control of data taking runs, plus fine-grained control of the O^2/FLP cluster.

The primary duty of a control mechanism for the ALICE O^2 system is to launch, configure and control a set of data-driven workflows inside a computer cluster. On top of this cluster control role, AliECS is in charge of

1. managing the lifetime of thousands of processes in the O^2/FLP cluster (while delegating control of O^2/EPN processes to a specialized control mechanism for the O^2/EPN cluster),
2. minimizing the waste of beam time by reusing processes and avoiding time-consuming process restart operations,
3. and interfacing with the LHC, the trigger system, the DCS (Detector Control System) [7] and other systems through common APIs.

Synchronous and Asynchronous Workflows

From a cluster control point of view, the primary task of AliECS is to handle the details of synchronous workflows, as this kind of workflow is time-critical and directly affected by experiment operations. Asynchronous workflows will be executed in Grid-like environments, both on the WLCG and inside the O^2 facility when resources are available, on a best-effort basis. At times when the O^2/EPN facility has free resources (i.e., compute resources not used for synchronous operation), the O^2/EPN control mechanism will have the responsibility of setting up asynchronous processing environments for tasks like asynchronous reconstruction, analysis, and simulation. AliECS will be able to reclaim resources assigned to asynchronous operation if synchronous processing workflows require them by requesting from the O^2/EPN to ensure immediate availability of these resources for synchronous operation.

AliECS DESIGN OVERVIEW

Due to the tight coupling required between high-level experiment control and O^2/FLP cluster control, AliECS integrates both control levels (experiment control and O^2/FLP cluster control) into a single system. Thus, AliECS provides in-depth control of every data-driven process running in the O^2/FLP cluster. It is foreseen for the AliECS core to further interface with the O^2/EPN control mechanism, but only for coarse-grained, high-level control of the O^2/EPN cluster.

AliECS is a distributed system in charge of the O^2 facility (directly or indirectly), with full knowledge and control over the resources of the O^2/FLP cluster. It implements a reliable and distributed state machine mechanism to represent the aggregated state of the constituent O^2 processes of a data-driven workflow. Furthermore, it allows reconfiguration and reuse of running O^2 processes as often as possible to avoid process restarts, and it allows simultaneous operation of multiple workflows, with easy reallocation of resources among workflows. Finally, it reacts promptly to inputs, handling events from the user, the LHC, the trigger system, the DCS, and the cluster itself with a high degree of autonomy.

The O^2 project has chosen FairMQ [8] as the common message passing and data transport framework for its data-driven processes. It has been developed in the context of FairRoot [9, 10], a simulation, reconstruction and analysis framework for particle physics experiments. FairMQ provides the basic building blocks to implement complex data processing workflows, including a message queue, a configuration mechanism, a state machine, and a plugin system.

Resource Management in the O^2/FLP Facility

We implement AliECS as a distributed application, using Apache Mesos [11, 12] as toolkit. This custom solution integrates a task scheduler component, a purpose-built distributed state machine system, a multi-source stateful process configuration mechanism, and a control plugin and library compatible with any data-driven O^2 process.

An Overview of Apache Mesos Apache Mesos is a cluster resource management system. It greatly streamlines distributed application development by providing a unified distributed execution environment. Mesos facilitates the

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-WEDPL02

management of O^2/FLP components, resources and tasks inside the O^2/FLP facility, effectively enabling the developer to program against the datacenter (i.e., the O^2/FLP facility at LHC Point 2) as if it was a single pool of resources.

Apache Mesos comes with two main components: masters and agents. In a Mesos-enabled cluster there is a Mesos agent running on every node: its purpose is to collect information on the resources available on that node, and to handle task deployment. A Mesos-enabled cluster must also have at least one Mesos master. In this context, a Mesos-aware distributed application is called a *framework*. When developing a framework, the developer must build a *scheduler* process (which subscribes to the Mesos master), as well as one or more *executors*. The Mesos master acts as an authoritative source of knowledge on cluster resources, and periodically sends resource offers to the schedulers of the frameworks running on the cluster, which can then use these resources to run tasks.

In order to run a task, a Mesos agent runs the selected executor component of the framework that accepted the resources provided by this agent, and the executor can then run a process or perform any other operation as required by the scheduler.

The Role of Apache Mesos in the O^2/FLP Facility

Apache Mesos has become a household name in the industry, and it has been used in deployments of 10,000s of nodes. It is an open source project, hosted by the Apache Software Foundation. Commercial support is available.

For AliECS, benefits of using Mesos include

1. the knowledge of what runs where,
2. resource management, which facilitates various deployment steps including port assignment, node selection, configuration, and others,
3. transport facilities for O^2-specific control messages,
4. task status tracking (e.g. an event is raised if a task dies unexpectedly),
5. and advanced features such as node attributes, resource overprovisioning, checkpointing, and others.

The drawback of having Apache Mesos as an additional component in the stack is compensated by its benefits. We also argue that implementing a computing system at the scale of O^2 with modern techniques would in any case involve a resource management system component or mechanism.

It is important to note that Apache Mesos is not a control system. The requirements, and thus the design of AliECS include much beyond Mesos.

AliECS Components

AliECS, our proposed solution for the problem of O^2/FLP synchronous control and ECS is under development. The current implementation of AliECS can be found on GitHub [13], and it consists of

1. the AliECS core (which includes the Apache Mesos-facing scheduler component),
2. the AliECS executor,

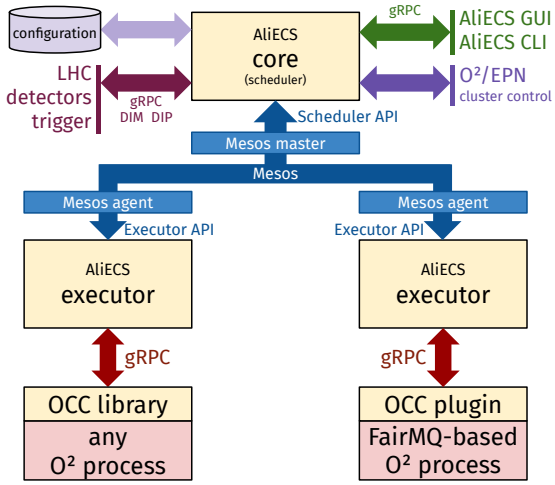

Figure 2: The AliECS architecture. All control communications between core and executor instances are piggybacked on Mesos messages. RPC-pattern interaction between the user interfaces and the AliECS core, and between the executor and the controlled O^2 process is implemented with gRPC. The OCC plugin hides the complexities of handling gRPC connections and driving the state machine.

3. the O^2 control and configuration plugin for FairMQ devices (OCC plugin),
4. the O^2 control and configuration library (OCC library),
5. the AliECS control and configuration command line utility (`coconut`),
6. the AliECS process execution and control utility for OCC library based O^2 processes (`peanut`),
7. and the web-based AliECS GUI.

The AliECS core accepts requests from the AliECS GUI or from `coconut`. These requests are then processed, and they result in Mesos API calls, handlers for Mesos API events, or O^2-specific control messages (using Mesos API calls and handlers for transport).

Furthermore, AliECS interfaces via a configuration wrapper library with Consul [14], a key-value store which acts as the system's configuration repository. The design also includes interfacing with information sources from the LHC, the trigger system, and the DCS.

Most components of AliECS are written in Go [15], a statically typed general purpose programming language in the tradition of C, which is particularly suitable for distributed system development because of its advanced synchronization and threading facilities. The OCC plugin is developed in C++17, and works with any FairMQ-based process. A non-plugin library equivalent of the latter is also provided, for O^2 processes which do not support the FairMQ plugin system.

Inter-process Communication in AliECS

The common idiom of inter-process communication in AliECS is gRPC [16], an open source, cross-language RPC (remote procedure call) system backed by Google. It is

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-WEDPL02

widely used in the microservices community. gRPC comes with a code generator which produces client and server stub code based on a common descriptor file. Once the developer describes the client-server interface in this file, the generator can output stubs for C++, Go or any other supported language, enabling seamless interaction between processes in a heterogeneous cross-language distributed system. Depending on the language, for the developer this gRPC-mediated remote interaction mimics local function calls.

In AliECS, gRPC is used for communication between the core and the user interface (both `coconut` and the web-based GUI), and for communication between the executor and the OCC plugin or library (see Fig. 2). gRPC interfaces are also under development for interaction with other systems such as the trigger system and the DCS.

Alternatives to gRPC were considered, such as a REST API. This approach would have required more effort on the client side, and it would have been less comfortable to use, especially for two way interaction between the executor and the OCC component. A framework such as Swagger [17] could mitigate some of these drawbacks. Several competitors to gRPC in the RPC space were also considered, including Cap'n Proto [18], MessagePack-RPC [19], JSON-RPC [20] and the `net/rpc` [21] package from the Go standard library, but most of them fell short on some key criteria, such as performance, developer support, suitability for cross-language communication, and ease of use.

AliECS Concepts

The basic unit of scheduling in AliECS is a *task*. A task generally corresponds to a process, specifically a process that can receive and respond to OCC-compatible control messages. All AliECS workflows are collections of tasks, which together form a coherent data processing chain.

Tasks are the leaves in a tree of *roles*. A role is a runtime subdivision of the complete system, it represents a kind of operation along with its resources. Each task implements one or more roles. Roles allow binding tasks or groups of tasks to specific host attributes, detectors and configuration values. Each role represents either a single task, or a group of child roles. If tasks are leaves, roles are all the other nodes in the control tree of an environment.

In comparison with the ECS partitions used in Run 2, we aim to provide novel, more flexible, and more easily deployable abstractions. In memory, a tree of O^2 roles, along with their tasks and their configuration is a *workflow*. A workflow aggregates the collective state of its constituent O^2 roles. A running workflow, along with associated detectors and other hardware and software resources required for experiment operation constitutes an *environment*.

The Environment State Machine Every environment has a distributed state machine, which drives the state of its tasks (see Fig. 3). In memory, each role as a node in the control tree also has a state, which aggregates the states of its child roles (or of its single child task). Thus, the state machines of each individual process are directed by the top-level state machine of the environment.

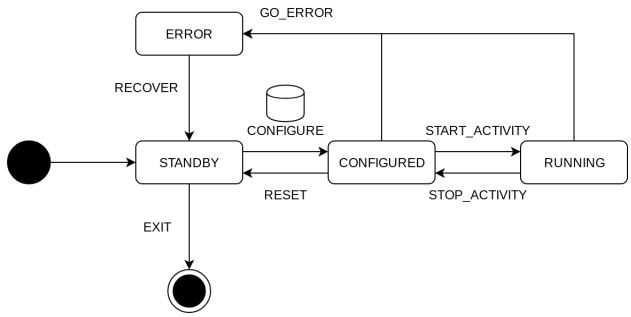

Figure 3: The state machine of an AliECS environment. The same state machine is implemented by each task. For FairMQ-based tasks the OCC plugin acts as a translation layer between the AliECS task state machine and the underlying FairMQ state machine.

When an environment is in state `"RUNNING"`, it implements an *activity*, such as a data taking run. Every run (or activity) has a unique run number, thus an environment can acquire multiple run numbers in its lifetime, for multiple runs, one at a time, with each run number only being valid during the `RUNNING` state, between a `"START_ACTIVITY"` transition and the subsequent `"STOP_ACTIVITY"` transition.

The environment state machine is the entry point for all process control operations. Some examples of control requests at this level include creating a new environment by loading a workflow template from a configuration repository (which also instantiates the new environment's state machine), requesting a state transition for an environment, or destroying an environment.

Configuration Management

AliECS is both a producer and consumer of configuration data in the O^2/FLP cluster. There are 3 kinds of configuration information that AliECS deals with:

1. the AliECS core configuration,
2. the AliECS workflow configuration,
3. and the O^2 tasks configuration.

Core Configuration The AliECS core configuration is a flat list of read-only values which the AliECS core acquires on startup. This kind of configuration can be populated from a file, from command line parameters, and from environment variables (whatever the person who deploys the software prefers, and these configuration sources can be combined). Typical values that come from this configuration mechanism are the control port to use for incoming AliECS GUI or `coconut` connections, the URI of the Mesos master API, and path of the AliECS executor on controlled nodes. Once set, this kind of configuration data cannot change throughout the lifetime of the AliECS core process.

Workflow Configuration The AliECS workflow configuration is acquired by way of a configuration manager

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-WEDPL02

subsystem which uses Git repositories as a backend for file storage and versioning. Ultimately, AliECS workflow configuration data consists of task descriptor files and workflow template files, and these are arranged in repositories. An AliECS workflow configuration repository is a Git repository with two directories, `tasks` and `workflows`. Each of these directories contains a flat structure of YAML files: task descriptors and workflow templates.

Every task descriptor file is a YAML document which describes how to launch and control a single task, such as an O^2 data-driven process. This kind of file contains information such as the path to an executable, command line arguments, environment variables, state machine type (i.e. whether the process uses the FairMQ state machine or the AliECS native state machine), available bound channels (for incoming connections) etc.

Every workflow template file is a YAML document which describes the structure of a workflow of roles and (ultimately) tasks. This structure directly expresses the control tree, which defines the layout of the distributed state machine. The input processing mechanism which loads a workflow template into memory to generate a workflow (which is in turn associated with an environment) provides several facilities to aid in workflow definition, such as iterators (to generate a sequence of integer-numbered roles) and hierarchical channel references (to describe task-to-task data flow channels in terms of their relative locations in the workflow rather than static labels).

The workflow template format is not primarily intended as a data interchange format: it is rather the human-readable representation of a curated structure of tasks to be run by AliECS as an environment. The O^2 DPL (Data Processing Layer) [22], used by O^2 task developers as a framework for building stateful data-driven device topologies is capable of generating AliECS-compatible workflow templates, starting from its own process topology description mechanism.

YAML has been proven advantageous for this kind of configuration structure because of its expressiveness, because of the fact that it is easily readable and editable by a human, and because of the availability of mature serialization and deserialization facilities available for Go, C++ and other languages.

Due to the need for reproducibility and thorough documentation of experiment operations, the O^2 computing system includes a bookkeeping system under development, called Jiskefet (a different system with similar functionality was used in ALICE Runs 1 and 2). Workflow and task descriptors are critical configuration data on experiment operations, so it is necessary to persist either this data or a permanent, versioned reference to this data to the bookkeeping system. It is also of interest for experiment operations to be able to move back and forth between workflow configuration revisions, and to be able to easily test alternatives. The choice of Git as versioning and file storage mechanism was driven by a need for versioning and structure. As an alternative, a variety of database and key-value storage systems were considered, including MongoDB [23], CouchDB [24] and Consul. While their use might have made simple querying somewhat more straightforward, an additional layer would have been needed in order to provide versioning.

By using Git, a state of the art versioning mechanism is available out of the box, allowing users to manage their own workflow configuration repositories and branches. The directory structure of a workflow configuration repository is enforced by the AliECS core when the user adds a repository, and the AliECS API presents each repository as a stateless, read-only configuration source.

Workflow configuration is complemented by AliECS runtime variables, which can affect the loaded workflow and single tasks.

O^2 Tasks Configuration AliECS implements task configuration as a push operation, as opposed to a pull from the task itself. By implementing a push mechanism in AliECS, it is ensured that task configuration is a time-constrained event associated with a state transition, restricting the freedom of task developers to query a configuration repository at any time and thus inadvertently store hidden state information in the task in question. Every task configuration is delivered as a payload with the `CONFIGURE` transition event. This payload includes communication channel configuration (i.e. hosts and ports to connect or bind) plus an optional key-value map of application-specific configuration data. The latter comes from Consul, and its content can also be affected by AliECS runtime variables, much like workflow configuration.

The AliECS command line interface (`coconut`) allows O^2 task developers to import their application-specific configuration into Consul. An ad-hoc library within AliECS makes sure that configuration import operations validate the incoming data and place it within a pre-defined structure. In this way, all O^2 task configuration data is arranged in a tree, by component (i.e. the kind of task, such as "readout" or "quality-control"), configuration entry name, and timestamp. In this mechanism, each component name identifies a kind of O^2 task, and each entry is a key which identifies an imported YAML, JSON, TOML or INI document, with the timestamp pointing to each of its revisions. The `coconut` interface also exposes some querying capabilities implemented in the AliECS core, which allow the user to browse the O^2 component configuration repository, including the modification history of every configuration entry.

By storing O^2 component configuration in Consul, we accept a limitation of 512 KB per configuration document. We however keep the option for application users and developers to include comments and formatting inside their configuration documents, and we make it possible for O^2 components to query Consul directly.

O^2 Process Control

Most O^2 processes are also FairMQ devices, i.e., programs that make use of the FairMQ library for its state machine and I/O facilities. Since DPL relies on FairMQ as underlying

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-WEDPL02

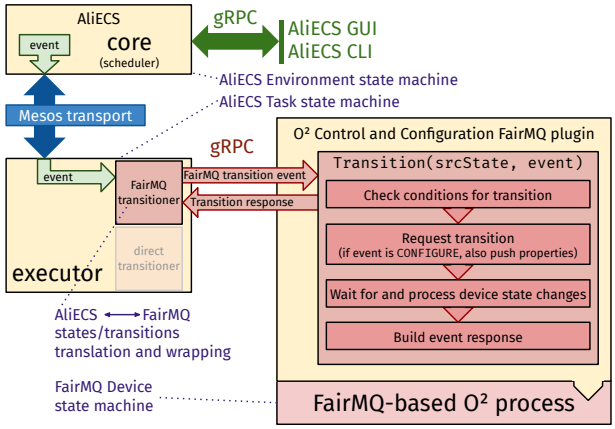

Figure 4: The AliECS executor integrates modular components called *transitioners*. These units act as translation wrappers between AliECS states and events, and the states and events of the state machine of a specific controlled process. In the figure above the executor has loaded the FairMQ transitioner, which drives the state machine of a FairMQ-based process.

framework, DPL devices are also FairMQ devices, which makes FairMQ a common denominator among O^2 tasks.

FairMQ provides a plugin system, which is capable of loading the purpose-built O^2 control and configuration plugin for FairMQ devices. This plugin enables any FairMQ device to accept control commands from an AliECS executor (see Fig. 4). The OCC plugin takes control of the process on startup, and starts a gRPC server on a specific TCP port as instructed by the AliECS executor. When the OCC plugin receives a remote procedure call from the executor, it drives the state machine of the FairMQ device and reports back. The OCC plugin is also capable of pushing configuration key-value pairs as FairMQ properties to the FairMQ configuration map of the device. This functionality is used for channel configuration, as well as for any other runtime value that needs to be pushed to a task, including run numbers and O^2 component specific configuration.

O^2 tasks are started on demand when the roles of an environment require them, and by default they are killed when their environment is disbanded. Optionally, upon environment shutdown the tasks can be kept running, and tracked in an *idle tasks pool*, ready to be reconfigured and used without additional deployment steps. Thus, since automatic port assignment and other crucial data flow setup operations happen at task configuration time rather than at task startup, it is possible to destroy an environment, modify the workflow template or components configuration, recreate an environment with some of the same task descriptors, and resume operation without having to redeploy all tasks.

AliECS IN RUN 3 DETECTOR COMMISSIONING

In order to facilitate deployment in ALICE Run 3 detector commissioning setups, an Ansible-based [25] installation system for AliECS and other O^2 components is being developed. This installation mechanism can be used via Foreman [26], a server lifecycle management solution, or through a custom command line installer tool for small-scale setups.

We expect to use these deployment tools throughout the O^2/FLP facility, and some of them have already been used for detector commissioning setups, including both single-node and multi-node AliECS instances.

AliECS instances for detector commissioning tasks include setups for ALICE detectors TPC, ITS and MFT. Some of the major challenges we encountered include:

1. the fact that AliECS is often deployed in an environment where ALICE detector teams already have their own tooling and scripts, which complicates integration;
2. the fact that IPC interfaces between AliECS and the DCS, and between AliECS and the trigger system aren't in place yet, which requires workarounds;
3. the fact that AliECS instances for detector teams need to be deployed either off-premises, or on-premises but in differently configured network environments, which complicates automation and support intervention;
4. the fact that AliECS integrates with a multitude of O^2 components, which makes integration testing critical for successful releases.

We have promptly reacted to these challenges by collecting further requirements from detector teams, and by more clearly communicating the potential integration points between detector team tooling and AliECS components. We have extended and improved `coconut`, which can easily be called within shell scripts to direct AliECS behavior, and we have engaged to extend AliECS so it can also execute generic commands, as opposed to only stateful OCC-compatible tasks.

We have further extended and refined our Foreman-based system configuration management facilities, and we have started work on a new Ansible-based multi-node installer system written in Go, as a replacement for the previously used wrapper script. A high level testing mechanism for the Ansible roles which install AliECS and other O^2 components was also developed, in order to spot integration issues as early as possible.

CONCLUSION

We propose a new, custom built, microservices oriented, integrated solution for ALICE experiment control as well as for cluster control in the FLP facility of the O^2 computing system. We assert that the leap to O^2 is an opportunity for a broad technical refresh by leveraging modern cluster resource management and IPC technologies for a high performance, low latency ECS.

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-WEDPL02

By taking advantage of Apache Mesos, we gain resource management, control message transport, events, and more, with the goal of achieving improved operational flexibility. On top of this framework, we implement a distributed state machine mechanism, with an expressive configuration format and a modular process control stack for maximum compatibility in an inevitably heterogeneous context. We employ open source cross-platform and cross-language technologies such as gRPC, Git and Consul to maximize interoperability and minimize technical risk.

We aim to maximize the usage of LHC beam time while ensuring optimal resource allocation in the new O^2 facility for both synchronous and asynchronous data-driven workflows. AliECS takes direct control over the O^2/FLP facility, and interfaces with the O^2/EPN cluster control to gain high-level oversiught of the whole data readout chain. With our design approach we aim to achieve substantial performance improvements and operational benefits in mission critical use cases compared to the previous system.

REFERENCES

[1] K. Aamodt *et al.*, "The ALICE experiment at the CERN LHC," *JINST*, vol. 3.08, S08002, 2008. DOI: 10.1088/1748-0221/3/08/S08002.

[2] B. Abelev *et al.*, "Upgrade of the ALICE Experiment: Letter Of Intent," *Journal of Physics G: Nuclear and Particle Physics*, vol. 41, no. 8, p. 087 001, 2014. DOI: 10.1088/0954-3899/41/8/087001.

[3] J. Adam *et al.*, "Technical Design Report for the Upgrade of the Online-Offline Computing System," CERN, Tech. Rep. CERN-LHCC-2015-006 / ALICE-TDR-019, 2015.

[4] J. Mitra, S. Khan, S. Mukherjee, and R. Paul, "Common Readout Unit (CRU) - A new readout architecture for the ALICE experiment," *Journal of Instrumentation*, vol. 11, no. 03, p. C03021, 2016. DOI: 10.1088/1748-0221/11/03/C03021.

[5] A. Borga *et al.*, "The C-RORC PCIe card and its application in the ALICE and ATLAS experiments," *Journal of Instrumentation*, vol. 10, no. 02, p. C02022, 2015. DOI: 10.1088/1748-0221/10/02/C02022.

[6] F. Carena *et al.*, "The ALICE data acquisition system," *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 741, pp. 130–162, 2014, ISSN: 0168-9002. DOI: 10.1016/j.nima.2013.12.015.

[7] P. Chochula *et al.*, "Challenges of the ALICE Detector Control System for the LHC RUN3," *Proceedings of the 16th International Conference on Accelerator and Large Experimental Control Systems*, International Conference on Accelerator and Large Experimental Control Systems, no. 16, pp. 323–327, Jan. 2018. DOI: 10.18429/JACoW-ICALEPCS2017-TUMPL09.

[8] *FairMQ C++ Message Queuing Library and Framework*, https://github.com/FairRootGroup/FairMQ, Accessed: 2019-09-26.

[9] M. Al-Turany *et al.*, "The FairRoot framework," *Journal of Physics: Conference Series*, vol. 396, p. 022 001, 2012. DOI: 10.1088/1742-6596/396/2/022001.

[10] M. Al-Turany *et al.*, "ALFA: The new ALICE-FAIR software framework," *Journal of Physics: Conference Series*, vol. 664, no. 7, p. 072 001, 2015. DOI: 10.1088/1742-6596/664/7/072001.

[11] *Apache Mesos*, http://mesos.apache.org/, Accessed: 2019-09-26.

[12] D. Berzano, G. Eulisse, C. Grigoraş, and K. Napoli, "Experiences with the ALICE Mesos infrastructure," *Journal of Physics: Conference Series*, vol. 898, no. 8, p. 082 043, 2017. DOI: 10.1088/1742-6596/898/8/082043.

[13] *The O^2 Control System*, https://github.com/AliceO2Group/Control, Accessed: 2019-09-26.

[14] *Consul by HashiCorp*, https://www.consul.io/, Accessed: 2019-09-26.

[15] *The Go Programming Language*, https://golang.org/, Accessed: 2019-09-26.

[16] *gRPC A high performance, open-source universal RPC framework*, https://grpc.io/, Accessed: 2019-09-26.

[17] *Swagger*, https://swagger.io/, Accessed: 2019-09-26.

[18] *Cap'n Proto*, https://capnproto.org/, Accessed: 2019-09-18.

[19] *MessagePack-RPC*, http://msgpack.org/rpc/rdoc/current/MessagePack/RPC.html, Accessed: 2019-09-18.

[20] *JSON-RPC 2.0 Specification*, https://www.jsonrpc.org/specification, Accessed: 2019-09-18.

[21] *Go Package rpc*, https://golang.org/pkg/net/rpc/, Accessed: 2019-09-18.

[22] *ALICE O^2 software*, https://github.com/AliceO2Group/AliceO2, Accessed: 2019-09-26.

[23] *mongoDB*, https://www.mongodb.com, Accessed: 2019-09-18.

[24] *Apache CouchDB*, http://couchdb.apache.org/, Accessed: 2019-09-18.

[25] *Red Hat Ansible*, https://www.ansible.com/, Accessed: 2019-09-19.

[26] *Foreman*, https://theforeman.org/, Accessed: 2019-09-19.