
Scalable Computing: Practice and Experience, ISSN 1895-1767, http://www.scpe.org
© 2024 SCPE. Volume 25, Issues 1, pp. 607–613, DOI 10.12694/scpe.v25i1.2970

CONFIGURATION OF CONTAINER DEPLOYMENTS ON THE COMPUTE CONTINUUM USING ALIEN4CLOUD

ADRIAN SPĂTARU* AND JULEN APERRIBAY†

Abstract. This paper presents the drawbacks and benefits of using Alien4Cloud as a platform for deploying container-based applications on the Compute Continuum. To achieve this, a plugin has been developed to deploy container-based applications in multiple Kubernetes clusters and to configure the containers based on their dependencies. More specifically, our implementation has been validated using a system of two Kubernetes clusters (one in Romania, and one in Spain) and a machine-learning application that has been successfully deployed using this system.

Key words: Compute Continuum, Service Orchestration

AMS subject classifications. 68M14, 68M15

1. Introduction. Alien4Cloud [9] is a mature platform built for the Development and Operations (DevOps) paradigm. It uses TOSCA (Topology Specification for Cloud Applications), an OASIS specification for modelling a complete application stack and automating its deployment and management in Cloud environments [6]. Using TOSCA, the administrator uploads component specifications and uses these definitions to create an application topology.

Alien4Cloud (A4C) offers an easy-to-use interface to Drag&Drop TOSCA components and link requirements of one with the capabilities of another. Nevertheless, this is not the main feature of the platform. Some components can register as services in the A4C Service Registry, which allows other applications to connect to these registered services. An example is explained in detail in Section3.2. Moreover, the platform's functionality can be extended through plugins. An Orchestrator plugin is responsible for deploying and monitoring an application topology, and is able to configure the parameters of the components before they start.

We use an extension of TOSCA, namely *TUFA* [13] and implement an Orchestrator plugin to manage Kubernetes deployments for the Container definitions in the TUFA specification. We validate our plugin using an industrial application that detects anomalies in the manufacturing process.

The key contributions of this paper are:

- the implementation of the Orchestrator plugin that transforms TUFA specifications in Kubernetes descriptions and interacts with the Kubernetes Orchestrator to deploy and monitor applications.
- the validation of the developed plugin using two Kubernetes clusters and a machine learning application
- an experience-driven assessment of the benefits and drawbacks of using a platform like Alien4Cloud for the management of deployments in the Compute Continuum setting

The paper is structured as follows. Section 2 presents the work behind the popularity of the TOSCA specification and the alternatives to Alien4Cloud. Section 3 describes the system architecture and details the use case scenario. Section 4 presents the benefits and drawbacks of Alien4Cloud as a platform for developing future-proof Orchestration plugins. Finally, Section 5 draws the conclusion.

2. Related Work. Numerous specifications in specific domains address the modelling of Cloud infrastructure, services, relationships, and policies for monitoring and scaling. A comprehensive review examining Cloud Modelling Languages (CML), as documented by Bergmayr et al. [4], highlighted TOSCA as the predominant

*Department of Computer Science, West University of Timişoara, Romania (Corespondig author: adrian.spataru@e-uvt.ro).

†IDEKO, Spain (japerribay@ideko.es)

one. Although certain CMLs [2, 8] predate the TOSCA standard, and some emerged concurrently [1, 3, 5, 12], no new modelling languages appeared since 2014. Furthermore, extensions of existing CMLs aim to integrate with the TOSCA specification, resulting in a convergence of cloud modelling semantics.

With the rise of Fog and Edge computing, the TOSCA model has attracted various proposals for extensions, focusing on diverse models for these resources. One approach involves using these resources as a backend for serverless computing [14]. The serverless paradigm allows users to define the topology in terms of application fragments, which are distributed across the resource fabric using load-balancing techniques.

SODALITE [7, 11] employs a static service optimization process based on prior application runs using different configurations.

In the realm of deploying TOSCA-based applications, several alternatives stand out as viable options to Alien4Cloud. Cloudify[1] is an open-source orchestration platform compliant with TOSCA standards. It offers capabilities for modeling, deployment, and management across hybrid cloud environments. Heat[2], a component-based orchestration tool for OpenStack environments, utilizes TOSCA templates to describe the necessary infrastructure and services for applications.

The previously mentioned alternatives are mature developments that provide extreme granularity with respect to the configuration of Virtual Machines and infrastructure in general. Our approach focuses on the DevOps perspective which shifts fast to the development of microservices which run in Containers. The most popular container orchestration engines Kubernetes, which is a resource generally managed outside of the DevOps pipeline, by a system administrator.

An alternative solution is provided by the developers of Alien4Cloud, more specifically the Kubernetes plugin[3]. One drawback that we identified with this plugin is the fine granularity for modelling applications. The application developer is required to exceed his expertise and configure Kubernetes entities, which may not be possible for all application developers. Our approach simplifies this knowledge gap and aids the application developer in defining the relations between software components, not Kubernetes infrastructure.

The TUFA specification [13] has definitions for microservices packaged as containers, and this paper presents their integration with the Kubernetes orchestration engine and the use of A4C services for the automatic configuration of containers based on their dependencies.

3. System Architecture. Various components play essential roles in managing the Application life cycle, as depicted in Figure 3.1. The Alien4Cloud platform serves as the runtime environment, supplemented with TUFA definitions and the Orchestrator plugin. The **User Interface** facilitates service definition management and application deployments. Additionally, the **Components Repository** serves as the storage hub for *Component Definitions*, encompassing requirements and dependencies. The *Application Developer* uses this component repository to store such definitions, which subsequently aids the *Application Operator* in creating a new Application Topology from scratch or starting from one created by the developer in the **Topology Repository**.

The **Orchestrator plugin**[4] represents our main contribution and is developed using the Alien4Cloud plugin development guidelines[5]. The plugin allows for the creation of multiple Orchestrators, each connected to a corresponding Kubernetes cluster. The *fabric8*[6] java library is used for communicating with the Kubernetes clusters via HTTP REST. The **Service Registry** allows the administrator to register external services that provide different functionalities like storage services or message queuing services. The administrator authorises each Orchestrator to use a given service. If the service is internal to the Kubernetes Cluster where it runs, then the administrator will only allow the corresponding Orchestrator to use it. It makes little sense to allow other Orchestrators to use this because the deployed components cannot access the service. By default, no Orchestrator is authorised to use any of the services registered with the platform.

[1] https://docs.cloudify.co
[2] https://docs.openstack.org/heat
[3] https://github.com/alien4cloud/alien4cloud-kubernetes-plugin
[4] https://github.com/adispataru/tufa-a4c-orchestrator
[5] https://alien4cloud.github.io/#/developer_guide/plugin.html
[6] https://github.com/fabric8io/kubernetes-client

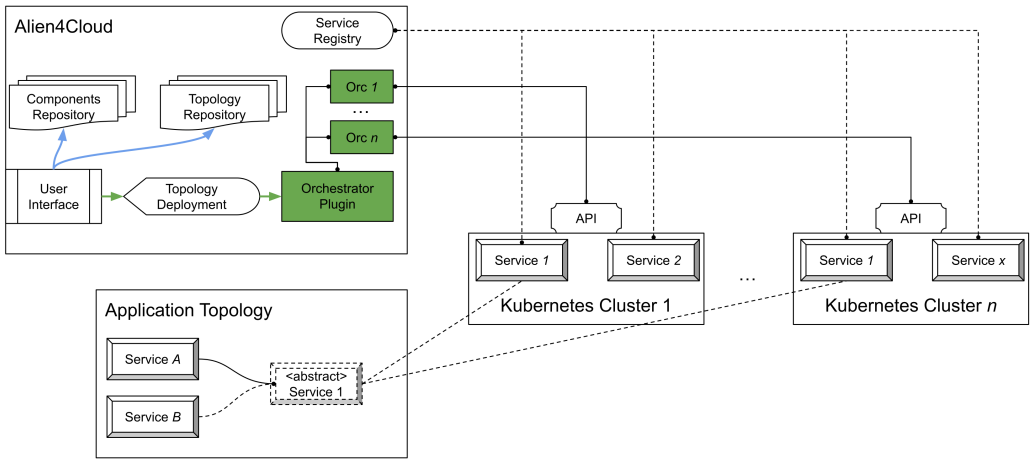

Fig. 3.1: System architecture

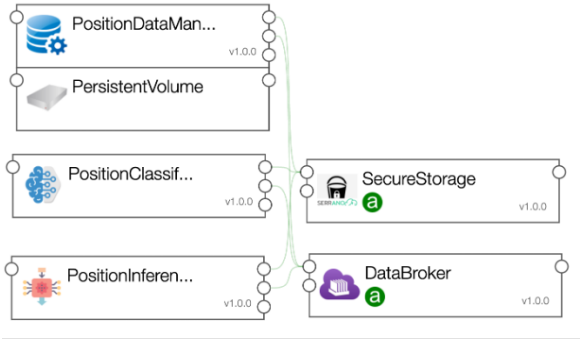

Fig. 3.2: Application definition using TUFA definitions in Alien4Cloud Topology Editor

3.1. Anomaly detection in manufacturing use-case. To better illustrate the advantages of the system, we present an example application. The application monitors data coming from a ball-screw component of a manufacturing machine and detects anomalies based on the position and acceleration of the ball-screw.

Figure 3.2 presents the application modelled using TUFA specification and viewed in the Alien4Cloud Topology Editor screen.

The application consists of 5 components:

- Data Manager component – responsible for locally persisting the data sent from the manufacturing machine, and uploading it to Secure Storage using the S3 interface. Additionally, this component monitors the accuracy of predicted anomalies. When the accuracy falls below a given threshold, the Data Manager uploads the local data to Secure Storage and contacts the Training component to use it and train a new model. This component uses a persistent volume to persist data locally and use it in case of recovery (e.g., container restart).
- Training component – responsible for training the machine-learning models and storing them using the S3 interface.
- Inference component – responsible for applying the trained machine learning model on new data.
- Data Broker – this component is an MQTT [10] service which directs messages. The industrial machine connects to this component and send data related to the position and acceleration of the ball-screw. The messages are redirected to the Data Manager component that schedules inference operations in

batch.

- Secure Storage – this component exposes an S3-compatible interface that allows users to store data. This component stores data used for training machine-learning models, the machine learning models used for inference, and the predicted anomalies.

All of the first three components (Data Manager, Training and Inference) depend on the other two for transferring messages and sending data. More concretely, the first three need to know the endpoint and credentials for routing messages and upload data. This configuration is achieved by binding two pieces of information: the *Config Map* data of a component and the inputs declared in the TUFA definition. More specifically, for the Data Manager component, the Config Map data consists of two YAML documents, one for configuring the application parameters and one for configuring the application logging mechanism. The training and inference component use similar configuration. An excerpt from the configuration file related to application parameters is presented in Listing 1.

Listing 1: Example definition of fields in the Config Map of the Data Manager component

```
mqtt:
  host: INPUT_DATABROKER_IP
  port: INPUT_DATABROKER_PORT
  keepalive: 60
  credentials:
    username: INPUT_DATABROKER_USER
    password: INPUT_DATABROKER_PASSWORD
storage:
  type: s3
  bucket_name: ideko-uc3-anomaly-detection
  credentials:
    gateway_url: INPUT_GATEWAY_URL
    skyflok_token: INPUT_GATEWAY_PASSWORD
```

The developer of the component will need to declare the values that require replacement (e.g. *INPUT_DATABROKER_IP*) as inputs of the *create* operation of the TUFA definition. An example is presented in Listing 2 for the Training component, but the Data Manager and the Inference components have the same inputs. The three only differ in the implementation chosen for creating the container, more specifically, the docker image used for deployment.

Listing 2: Example definition of inputs for the fields present in the Config Map of the Training component

```
interfaces:
  Standard:
    create:
      inputs:
        INPUT_DATABROKER_IP: { get_property: [REQ_TARGET, mqtt, ip_address]}
        INPUT_DATABROKER_PORT: { get_property: [REQ_TARGET, mqtt, port]}
        INPUT_DATABROKER_USER: { get_property: [REQ_TARGET, mqtt, user]}
        INPUT_DATABROKER_PASSWORD: { get_property: [REQ_TARGET, mqtt, password]}
        INPUT_GATEWAY_URL: { get_property: [REQ_TARGET, storage, url]}
        INPUT_GATEWAY_PASSWORD: { get_property: [REQ_TARGET, storage, password]}
      implementation:
        file: serrano/acceleration-classifier-training:0.1
        repository: serrano
        type: tufa.art.Deployment.Image.Container
```

For the *INPUT_DATABROKER_IP* parameter, the *get_property* function receives 3 arguments:

1. *REQ_TARGET* specifies that the property is to be examined in the target which satisfies a requirement of the current component
2. *mqtt* specifies the name of requirement
3. *ip_address* specifies the name of the property found in the requirement target.

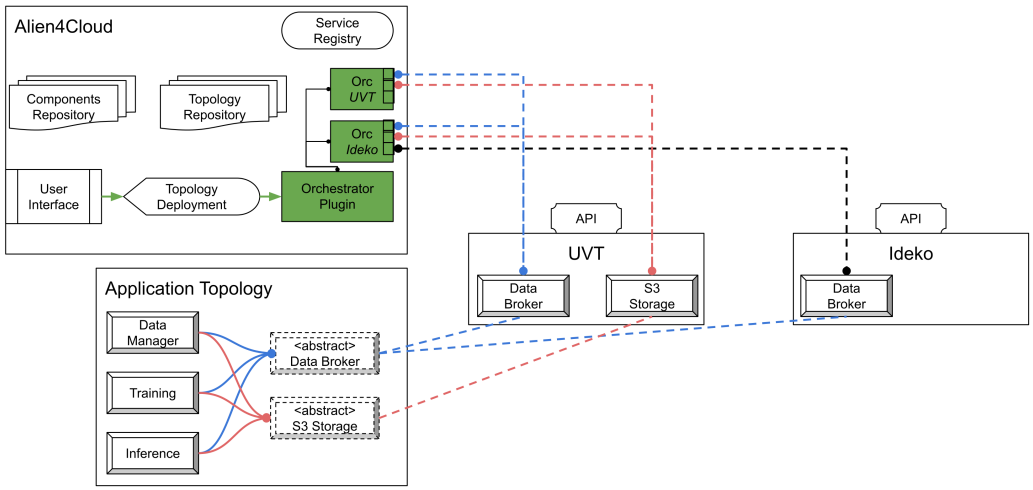

Fig. 3.3: System configuration for the validation of the plugin

The target that satisfies the *mqtt* requirement is the Data Broker component. Therefore, the IP address of the Data Broker will replace the *INPUT_DATABROKER_IP* text in the Config Map of the corresponding service. To achieve this during run-time, the dependency (i.e. Data Broker) is deployed before the dependent component (e.g. Training component) such that the IP of the dependency is known and can be injected in the configuration of the dependent component. However, the Data Broker and Secure Storage services presented in Figure 3.2 are abstract, and the Alien4Cloud platform will find a match for them using the Service Registry.

It is essential to observe the green letter "a" that appears in the boxes corresponding to the Data Broker and Secure Storage services from Figure 3.2. The green letter "a" means that the two services are abstract: they don't need to be deployed together with the other three components. Rather, they should be *matched* with Services that have been registered with Alien4Cloud.

3.2. System under test. For a more comprehensive understanding of the mechanism that matches components, we present the system under test. We have installed Alien4Cloud and have uploaded the Orchestrator plugin and the TUFA specifications for the use case.

Figure 3.3 offers a concrete view on the system. Two Kubernetes clusters are registered the with the Orchestrator. One Kubernetes cluster is installed at a university, Universitatea de Vest din Timisoara (UVT), in the west of Romania, consisting of three workers using HPE ProLiant DL385 Gen10 hardware. The second cluster is installed at a company, IDEKO, in the north of Spain, and consists of 3 Edge devices developed by IDEKO and installed on milling machines. The specific performance of the clusters is not the scope of this paper since performance of the application is not under scrutiny. Nevertheless, it's worth mentioning that an IDEKO worker has 4 cores and 8 GB of RAM, a pretty powerful edge device. The UVT worker has four times the number of cores and eight times the size of RAM, framing this as a Fog infrastructure provider.

There are two Orchestrator instances, one for UVT and one for IDEKO. In the UVT cluster, we have the two services that our use-case application depends on: the Data Broker and the Storage Service. The services have public access based on authentication and thus both Orchestrators have access to the two services.

The Data Broker instance deployed on Ideko premises also uses an authentication mechanism but does not have a public IP, thus only containers deployed on the Ideko Kubernetes cluster can reach it. In this case, only the Orchestrator instance corresponding to the Ideko cluster is authorised by the Alien4Cloud administrator to use the Ideko Data Broker.

3.3. Application Deployment. After using the Topology Editor as presented in Figure 3.2, the Alien4-Cloud interface leads the user through several decisions before deployment, out of which two are of interest:

1. Locations – The user needs to select the location for the deployment of the application (i.e., UVT or

Table 3.1: Configurations and functional tests

Test	Explanation
Container Image	Container image can be downloaded from a private repository. If the credentials for the repository do not exist in the Kubernetes cluster, they are created.
Persistent Volume Claim creation	Persistent Volume Claims (PVC) can be created. In the case of UVT, persistent volume claims are managed dynamically by a storage class. In the case of Ideko, local Persistent Volumes are created by our plugin apriori to satisfy the PVC.
Container binding to PVC	If a component requests a volume attachment, then the PVC is created and bound to the container at start-up
Config Map configuration	The Config Map has been successfully updated to replace the placeholder information with actual IP and credentials for the services matched in Alien4Cloud.
Matching services connection	The deployed components can access and interact with the services selected during the matching process.
Cleanup	All deployments, Config Maps and ephemeral volume claims are deleted from the Kubernetes cluster.

Ideko)

2. Matching – Abstract components are matched against the Service Registry; the user needs to select which service to use for each corresponding abstract component

Depending on the selected location, Alien4Cloud filters out matching nodes. In our experiment, if the user chooses the UVT location, the only options are to use both the Data Broker and the Storage services deployed at UVT. If the user selects the Ideko location, then both the UVT and Ideko instances of the Data Broker are available for the user to choose from. The user will select the Ideko instance to speed up the communication between these components. However, for the Storage service no instance is available on the Ideko premises. Thus, the components will use the instance deployed at UVT (since it is publicly available).

Finally, the application has been successfully deployed using all possible configurations and all functional tests presented in Table 3.1 have been passed.

4. Discussion. During the development of the plugin, we have encountered many circumstances, and we reckon the reader is interested in the details. This section presents the drawbacks and benefits of using Alien4Cloud to deploy applications on the Compute Continuum.

Benefits:

- Fit for purpose – in the context of the current investigation, Alien4Cloud solved an important problem in our use-case, most importantly the matching of external services
- Easy development – The plugin development process is relatively easy for an user which has experience with TOSCA and Java syntax. Documentation is accessible and well written.
- Built-in authorisation – The platform provides a complex mechanism for authorisation based on users and groups, handling access to all resources: applications, services, locations.

Drawbacks:

- No official maintenance – during the development of our plugin, the developers of Alien4Cloud announced the final software release with no future maintenance.
- Hard maintenance – the platform has been originally developed using Java 1.8, and several soft breaking changes have been introduced since the introduction of modules in Java 1.9. The breaking changes have not been solved until the final release, and the most recent Java version that can be used to compile a plugin for the platform is version 1.15.
- Design limitations – Some limitations are inherited from the original purpose of deploying applications in Virtual Machines. This did not impose any limits for deploying Containers, but future standards can not be accommodated since maintenance reached its end.

Overall, the platform offers the intended functionality with few efforts.We successfully extended its functionality through TOSCA based definitions and our Orchestrator plugin. The extended functionality allowed

us to achieve our goal of deploying containers in two locations on the Compute Continuum: Edge and Fog.

However, the end of official maintenance for the Alien4Cloud project imposes hard effort to invest in migrating the codebase to a more recent version of Java.

5. Conclusion. This paper investigated the suitability of the Alien4Cloud platform in managing the configuration of container deployments on the Compute Continuum. In oder to achieve our goal, the TUFA extension to the TOSCA specification has been used to define container-based applications, and an Orchestrator plugin has been developed to interact with the Kubernetes API in order to deploy an application topology.

Our experimental setup involved two Kubernetes clusters, one with higher performance and internet bandwidth, considered a Fog node, and one with lower performance, consisting of three Edge devices. A machine learning application for anomaly detection in the manufacturing industry has been successfully been deployed to both clusters. Moreover, the components of an application can access services deployed across clusters with the help of the Orchestrator plugin developed.

If an official maintainer is found for the Alien4Cloud platform, it can be an important piece in the puzzle of deploying applications on the Compute Continuum.

Acknowledgement. This work was partially supported by a grant of the Romanian Ministry of Education and Research, CNCS - UEFISCDI, project number PN-III-P4-ID-PCE-2020-0407, within PNCDI III. The work has been partially supported by a grant of European Union's Horizon 2020 Research and Innovation programme under grant agreement No 101017168, acronym SERRANO.

REFERENCES

[1] V. Andrikopoulos, A. Reuter, S. Gómez Sáez, and F. Leymann, *A gentl approach for cloud application topologies*, in European Conference on Service-Oriented and Cloud Computing, Springer, 2014, pp. 148–159.

[2] D. Ardagna, E. Di Nitto, P. Mohagheghi, S. Mosser, C. Ballagny, F. D'Andria, G. Casale, P. Matthews, C.-S. Nechifor, D. Petcu, et al., *Modaclouds: A model-driven approach for the design and execution of applications on multiple clouds*, in 2012 4th International Workshop on Modeling in Software Engineering (MISE), IEEE, 2012, pp. 50–56.

[3] T. Benson, A. Akella, A. Shaikh, and S. Sahu, *Cloudnaas: a cloud networking platform for enterprise applications*, in Proceedings of the 2nd ACM Symposium on Cloud Computing, 2011, pp. 1–13.

[4] A. Bergmayr, U. Breitenbücher, N. Ferry, A. Rossini, A. Solberg, M. Wimmer, G. Kappel, and F. Leymann, *A systematic review of cloud modeling languages*, ACM Computing Surveys (CSUR), 51 (2018), pp. 1–38.

[5] A. Bergmayr, J. Troya Castilla, P. Neubauer, M. Wimmer, and G. Kappel, *Uml-based cloud application modeling with libraries, profiles, and templates*, in CloudMDE 2014: 2nd International Workshop on Model-Driven Engineering on and for the Cloud co-located with the 17th International Conference on Model Driven Engineering Languages and Systems (MoDELS 2014)(2014), p 56-65, CEUR-WS, 2014.

[6] T. Binz, U. Breitenbücher, O. Kopp, and F. Leymann, *Tosca: portable automated deployment and management of cloud applications*, in Advanced Web Services, Springer, 2014, pp. 527–549.

[7] E. Di Nitto, J. Gorroñogoitia, I. Kumara, G. Meditskos, D. Radolović, K. Sivalingam, and R. S. González, *An approach to support automated deployment of applications on heterogeneous cloud-hpc infrastructures*, in 2020 22nd International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), IEEE, 2020, pp. 133–140.

[8] X. Etchevers, T. Coupaye, F. Boyer, and N. De Palma, *Self-configuration of distributed applications in the cloud*, in 2011 IEEE 4th International Conference on Cloud Computing, IEEE, 2011, pp. 668–675.

[9] FastConnect, *Application lifecycle enablement for cloud*. online, 2016. Accessed July 25, 2017.

[10] U. Hunkeler, H. L. Truong, and A. Stanford-Clark, *Mqtt-s—a publish/subscribe protocol for wireless sensor networks*, in 2008 3rd International Conference on Communication Systems Software and Middleware and Workshops (COMSWARE'08), IEEE, 2008, pp. 791–798.

[11] I. Kumara, G. Quattrocchi, D. Tamburri, and W.-J. V. D. Heuvel, *Quality assurance of heterogeneous applications: The sodalite approach*, in European Conference on Service-Oriented and Cloud Computing, Springer, 2020, pp. 173–178.

[12] G. C. Silva, L. M. Rose, and R. Calinescu, *Cloud dsl: A language for supporting cloud portability by describing cloud entities.*, in CloudMDE@ MoDELS, 2014, pp. 36–45.

[13] A. Spătaru, G. Iuhasz, and S. Panica, *Tufa: A tosca extension for the specification of accelerator-aware applications in the cloud continuum*, in 2022 IEEE 46th Annual Computers, Software, and Applications Conference (COMPSAC), IEEE, 2022, pp. 1178–1183.

[14] A. Tsagkaropoulos, Y. Verginadis, M. Compastié, D. Apostolou, and G. Mentzas, *Extending tosca for edge and fog deployment support*, Electronics, 10 (2021).

Edited by: Dana Petcu
Research papers
Received: Dec 3, 2023
Accepted: Dec 19, 2023