Livrable 1: Semaine 5

* Introduction
* Présentation de l'équipe (rôles)
* Etude de l’existant (+ problématiques, objectifs)
* analyse de risques et faisabilité
* Etat de l’art (etude théorique)
* Planning et processus de travail
* Outils de communications

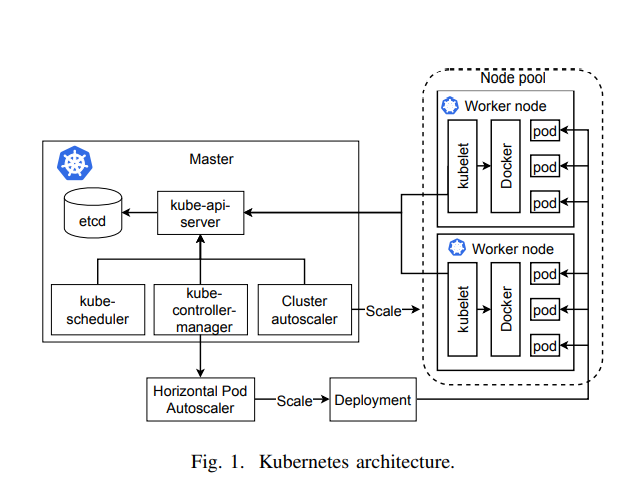
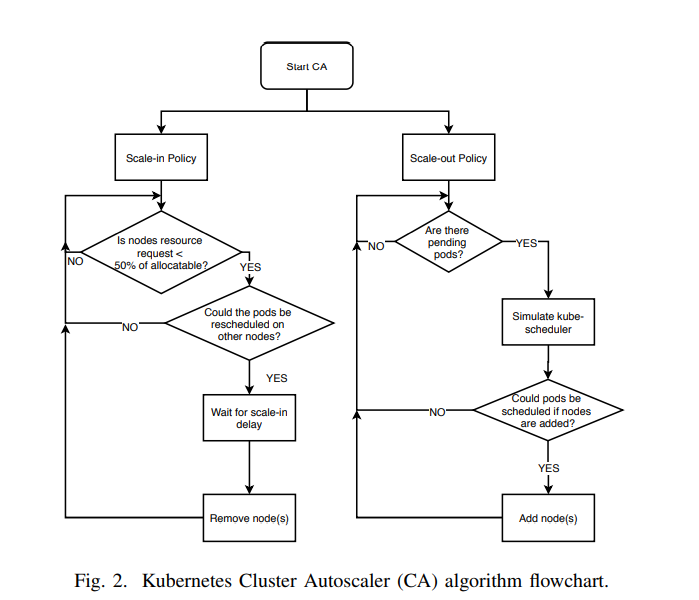
**Kubernetes:**

* There are many reasons why people come to use containers and container APIs like Kubernetes, but we believe they can all be traced back to one of these benefits:
* Velocity
* Scaling (of both software and teams)
* Abstracting your infrastructure
* Efficiency
* In this way, containers and Kubernetes can provide the tools that you need to move quickly, while staying available. The core concepts that enable this are:
* Immutability: In contrast, in an immutable system, rather than a series of incremental updates and changes, an entirely new, complete image is built, where the update simply replaces the entire image with the newer image in a single operation. There are no incremental changes. As you can imagine, this is a significant shift from the more traditional world of configuration management.
* Declarative configuration: ….the way you describe your application to Kubernetes. Everything in Kubernetes is a declarative configuration object that represents the desired state of the system. It is the job of Kubernetes to ensure that the actual state of the world matches this desired state.

* Online self-healing systems: Kubernetes is an online, self-healing system. When it receives a desired state configu‐ ration, it does not simply take a set of actions to make the current state match the desired state a single time. It continuously takes actions to ensure that the current state matches the desired state. This means that not only will Kubernetes initialize your system, but it will guard it against any failures or perturbations that might destabilize the system and affect reliability.
* **Scalability :** scalability by favoring decoupled architectures.
* Decoupling: In a decoupled architecture, each component is separated from other components by defined APIs and service load balancers. APIs and load balancers isolate each piece of the system from the others. APIs provide a buffer between implementer and con‐ sumer, and load balancers provide a buffer between running instances of each service.

Decoupling components via load balancers makes it easy to scale the programs that make up your service, because increasing the size (and therefore the capacity) of the program can be done without adjusting or reconfiguring any of the other layers of your service.

Decoupling servers via APIs makes it easier to scale the development teams because each team can focus on a single, smaller microservice with a comprehensible surface area. Crisp APIs between microservices limit the amount of cross-team communica‐ tion overhead required to build and deploy software. This communication overhead is often the major restricting factor when scaling teams.

* Kubernetes provides numerous abstractions and APIs that make it easier to build these decoupled microservice architectures:
* • Pods, or groups of containers, can group together container images developed by different teams into a single deployable unit.
* • Kubernetes services provide load balancing, naming, and discovery to isolate one microservice from another.
* • Namespaces provide isolation and access control, so that each microservice can control the degree to which other services interact with it.
* • Ingress objects provide an easy-to-use frontend that can combine multiple micro‐ services into a single externalized API surface area.
* Finally, decoupling the application container image and machine means that different microservices can collocate on the same machine without interfering with one another, reducing the overhead and cost of microservice architectures. The health checking and rollout features of Kubernetes guarantee a consistent approach to application rollout and reliability that ensures that a proliferation of microservice teams does not also result in a proliferation of different approaches to service production lifecycle and operations.

<https://drive.google.com/file/d/191JqlnH0jWAnQh3MGOXJKh9bdnDX8nUv/view?usp=sharing>